O'REILLY®



图灵程序设计丛书



Hadoop安全

大数据平台隐私保护

Hadoop Security: Protecting Your Big Data Platform

"Hadoop之父"推荐阅读

[美] Ben Spivey, Joey Echeverria 著

赵双 白波 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

译者介绍

赵双（DFlower）

Insight-Labs网络安全小组成员，中国科学院信息工程研究所助理研究员，中国科学院大学授课讲师。具有多年网络安全研究实战及大数据技术应用经验，研究方向包括恶意代码及APT检测、智能终端安全、大数据安全等，多次在XCON、OWASP、HITCON等国内外网络安全峰会发表技术演讲。

白波（Schnix）

毕业于西安交通大学，现任中国科学院信息工程研究所工程师。具有多年安全数据分析实战经验，研究方向包括APT检测分析、大数据安全、智能终端安全等，参与牵头起草行业标准1篇，获得省部级科学技术奖1次。

 图灵程序设计丛书

Hadoop安全：大数据平台隐私保护

Hadoop Security

Protection Your Big Data Platform

[美] Ben Spivey & Joey Echeverria 著

赵双 白波 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社

北 京

图书在版编目（CIP）数据

Hadoop安全 : 大数据平台隐私保护 / (美) 本•斯派维 (Ben Spivey) , (美) 乔伊•爱彻利维亚 (Joey Echeverria) 著 ; 赵双, 白波译. -- 北京 : 人民邮电出版社, 2017.9
（图灵程序设计丛书）
ISBN 978-7-115-46771-3

Ⅰ. ①H… Ⅱ. ①本… ②乔… ③赵… ④白… Ⅲ. ①数据处理软件 Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2017)第213821号

内 容 提 要

本书阐述了Hadoop从早期开放的消费互联网时代到现在作为敏感数据可信平台的演变历程,介绍了包括身份验证、加密、密钥管理和商业实践在内的诸多主题,并在实际环境下加以讨论。第1章是介绍性内容，随后分为四大部分：第一部分是安全架构，第二部分是验证、授权和安全审计，第三部分是数据安全，第四部分是归纳总结。最后介绍了几个使用案例，融合了书中诸多概念。

本书适合对Hadoop感兴趣的读者，有大数据平台保护需求的读者。

◆ 著　[美] Ben Spivey Joey Echeverria
译　赵 双 白 波
责任编辑　陈 曦
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编 100164　电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000 1/16
印张：16
字数：378千字　2017年9月第1版
印数：1－3 500册　2017年9月北京第1次印刷

著作权合同登记号　图字：01-2017-0539号

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

目录

第二部分 验证、授权和审计

第四部分　综合应用

序

近来，“Hadoop 安全”成为了一个充满矛盾的名词。雅虎和 Facebook 等公司创建和使用的这个大数据平台的早期版本，在保护其存储的数据方面并没有花费太多精力。实际上也没有必要这么做，因为 Hadoop 里几乎没有什么敏感数据，只有对黑客毫无吸引力的状态更新和新闻报道。而且，找到它们也不需要花费很多精力。

然而，当这种平台被更多传统企业使用时，便开始涉及更多传统企业数据。金融交易、个人银行账号和税务信息、医疗记录等诸如此类的信息，正是黑客们所追逐的。Hadoop 如今广泛用于零售、银行和医疗保健行业，它也逐渐引来了小偷们的注意。

如果说数据是有吸引力的目标，那么大数据的规模和吸引力都是最大的。与之前的任何预处理系统相比，Hadoop 能从更多地方收集更多数据，然后将它们结合起来，用更多方法进行分析。Hadoop 正是通过这种方式创造了巨大的价值。

显然，“Hadoop 安全”意义重大。

本书的两位作者在将安全概念引入 Hadoop 平台方面做出过突出贡献。他们在书中阐述了 Hadoop 从早期开放的消费互联网时代到现在作为敏感数据可信平台的演变历程，回顾了 Hadoop 安全的历史，既涵盖了其优点和演进过程，也涉及了新的业务问题。身份验证、加密、密钥管理和商业实践在内的诸多主题及其在实际环境下的应用也在书中有详细讨论。

10 年前，Hadoop 被 Facebook 选作图片存储软件，它发展到今天的过程非常有趣。如今，Hadoop 为数据处理和分析提供了更强的能力、更多的方法、更大的规模和更好的表现。因此，无论分开来看还是整体观之，Hadoop 也需要更多保护。

本书最好的地方在于，不仅仅对问题进行描述，还给出了处理建议。它能清晰并且详细地告诉读者，搭建和使用 Hadoop 的资深开发者们如何安全地管理大数据。本书给读者提供了如何使用——并且安全地使用——这个先进的平台以分析、处理和理解数据的最佳建议。

——Mike Olson

Cloudera 公司首席战略官、联合创始人

前言

虽然 Apache Hadoop 仍然是一个相对较新的技术，但这并没有限制它迅速地被业界采用，并爆发式地出现了很多相关工具，正是这些工具组成了 Hadoop 广阔的生态系统。对于 Hadoop 用户，这当然是个令人振奋的时代。Hadoop 给公司带来了前所未有的增值机会，同时也给负责数据访问安全和系统合规的技术人员带来了很多挑战。目前，已经有丰富的信息能够帮助使用 Hadoop 构建解决方案的开发者，以及部署运维 Hadoop 的管理员。然而，关于如何设计和实现 Hadoop 安全部署的指导信息依然匮乏。

本书全面而深入地介绍了 Hadoop 的众多安全特性，并使用通用的计算机安全概念进行组织。第 1 章是介绍性内容，随后分为 4 大部分：第一部分是安全架构，第二部分是验证、授权和安全审计，第三部分是数据安全，第四部分是归纳总结。这些部分涵盖了从设计安全架构的前期步骤，到实现通用的安全访问控制和数据保护。最后介绍了几个用例，融合了书中的很多概念。

目标读者

本书的目标人群是管理大数据平台安全的 Hadoop 管理员，以及需要在大型企业架构中设计集成 Hadoop 安全规划的安全架构师。书中介绍了很多 Hadoop 安全概念，包括验证、授权、审计、加密和系统架构。

第 1 章是对贯穿全书的一些安全概念的综述，以及对 Hadoop 生态系统的简介。如果刚刚接触 Hadoop，建议你阅读《Hadoop 技术详解》和《Hadoop 权威指南》。本书假设读者熟悉 Linux、计算机网络以及一般的系统架构知识。对于那些在保护分布式系统安全方面没有经验的管理员，第 2 章会给出这部分内容的概述。有经验的安全架构师如果不想回顾这方面知识，可以跳过这一章。总体而言，本书不要求读者具备编程背景，而尽量将内容重点集中于 Hadoop 安全在架构和操作方面的实现。

排版约定

本书使用以下排版约定。

- 楷体

  表示新术语和重点强调的内容。

- 等宽字体（`constant width`）

  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键词等。

- 加粗等宽字体（**`constant width bold`**）

  表示命令以及其他需要用户输入的文字。

- 等宽斜体（*`constant width italic`*）

  表示这些值应该替换为用户输入，或根据上下文确定。



该图标表示提示或建议。



该图标表示一般性说明。



该图标表示警告或警示。

使用代码示例

我们在全书提供了配置文件，用以指导读者为自己的 Hadoop 环境进行安全防护。这些样例中，可供下载的版本链接为：https://github.com/hadoop-security/examples。在第 13 章，本书提供了设计、实现和部署一个保存网页快照的网络接口的完整样例。该样例的完整源代码以及关于可部署应用的 Hadoop 集群的安全配置说明，都可以在 GitHub 上下载（https://github.com/hadoop-security/kite-spring-hbase-example）。

本书旨在帮助读者完成自己的工作。通常情况下，如果书中包含代码样例，你可以在自己的程序和文档中使用它们，不需要联系我们申请授权，除非需要直接复制相当一部分的代码。例如，编写程序时，使用本书中的几个代码片段并不需要申请授权，但出售或分发 O'Reilly 图书代码样例的光盘则需要获得许可；引用本书或引用样例解答问题不需要授权，但将本书的大量样例代码纳入产品的文档则需要获得许可。

我们不强制要求你在引用本书内容时进行声明，但如果你这么做，我们会非常感激。引用

信息通常包括书名、作者、出版社和 ISBN，如：*Hadoop Security* by Ben Spivey and Joey Echeverria (O'Reilly). Copyright 2015 Ben Spivey and Joey Echeverria, 978-1-491-90098-7。

若你认为对样例代码的使用需要授权，请通过这个邮箱联系我们：permissions@oreilly.com。

Safari® Books Online

Safari Books Online 是应运而生的数字图书馆，它同时以图书和视频的形式出版世界顶级技术和商业作家的专业作品。

技术专家、软件开发人员、Web 设计师、商务人士和创新专家等，都将 Safari Books Online 作为开展调研、解决问题、学习和认证培训的首选资源。

Safari Books Online 为企业、政府、教育和个人提供各种产品组合和灵活的定价策略。

会员可以通过搜索，从数据库中访问数以千计的图书、培训视频和正式出版前的书稿，这些数据来源包括 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 等。要了解 Safari Books Online 的更多信息，请访问我们的网站（http://www.safaribooksonline.com/）。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

我们为本书提供了专门网页，上面有勘误表、示例以及其他信息。可以通过 http://bit.ly/hadoop-security 访问该网页。本书中文版勘误可到 http://www.ituring.com.cn/book/1600 提交。

为本书提供建议或咨询技术问题，请发邮件到 bookquestions@oreilly.com。

想了解更多关于 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：
http://www.oreilly.com。

我们的其他联系方式如下：
Facebook：http://facebook.com/oreilly

Twitter：http://twitter.com/oreillymedia
YouTube：http://www.youtube.com/oreillymedia

致谢

Ben 和 Joey 要感谢以下所有人：编辑 Marie Beaugureau 和所有 O'Reilly 公司的工作人员；Ann Spencer；贡献了友情客串章节的 Eddie Garcia；主要技术审阅者 Patrick Angeles、Brian Burton、Sean Busbey、Mubashir Kazia 和 Alex Moundalexis；Jarek Jarcec Cecho；提供宝贵意见的其他作者 Eric Sammer、Lars George 和 Tom White；还有为我们提供集体支持的 Cloudera 的伙伴们，本书在他们的帮助下才得以问世。

来自Joey的致谢

我想将此书献给 Maria Antonia Fernandez、Jose Fernandez 和 Sarah Echeverria，他们在此前的每个日子里都鼓舞着我，让我觉得自己能完成想要完成的任何事情。我还要感谢我的父母 Maria 和 Fred Echeverria，我的兄弟姐妹 Fred、Marietta、Angeline、Paul Echeverria 和 Victoria Schandevel，他们始终给我爱与支持。此外，若是没有 Apache Hadoop 社区难以置信的强大支持，我无法完成本书。这里我无法列出帮助我们的每个人，不过 Ben 在后面列出了一些——当然不是全部。最后，我想感谢我的合著者 Ben：我们共同完成了一件了不起的事情，Bennie（Paul，别客气）。

来自Ben的致谢

我想以此书纪念 Ginny Venable 和 Rob Trosinski，我深深地怀念他们。我想感谢妻子 Theresa，是你给予我无限支持和理解，以及始终让我微笑的 Oliver Morton。感谢我的父母 Rich 和 Linda，谢谢你们长期以来向我展示教育的价值，并以卓越的专业才能树立了榜样。感谢 Matt、Jess、Noah，以及 Spivey 家庭的其他成员，感谢 Mary、Jarrod 和 Dolly Trosinsk，感谢 Swope 一家，以及整个过程中给予我极大帮助的朋友们：Hemal Kanani (BOOM)、Ted Malaska、Eric Driscoll、Paul Beduhn、Kari Neidigh、Jeremy Beard、Jeff Shmain、Marlo Carrillo、Joe Prosser、Jeff Holoman、Kevin O'Dell、Jean-Marc Spaggiari、Madhu Ganta、Linden Hillenbrand、Adam Smieszny、Benjamin Vera-Tudela、Prashant Sharma、Sekou Mckissick、Melissa Hueman、Adam Taylor、Kaufman Ng、Steve Ross、Prateek Rungta、Steve Totman、Ryan Blue、Susan Greslik、Todd Grayson、Woody Christy、Vini Varadharajan、Prasad Mujumdar、Aaron Myers、Phil Langdale、Phil Zeyliger、Brock Noland、Michael Ridley、Ryan Geno、Brian Schrameck、Michael Katzenellenbogen、Don Brown、Barry Hurry、Skip Smith、Sarah Stanger、Jason Hogue、Joe Wilcox、Allen Hsiao、Jason Trost、Greg Bednarski、Ray Scott、Mike Wilson、Doug Gardner、Peter Guerra、Josh Sullivan、Christine Mallick、Rick Whitford、Kurt Lorenz、Jason Nowlin 和 Chuck Wigelsworth。最后，我要感谢 Joey，还好他帮我一起写了这本书——我从不认为能够独自完成本书。如果我由于疏忽忘记了一些朋友，请接受我最诚挚的歉意。

来自Eddie的致谢

我想感谢我的家人和朋友，他们在我第一本书的写作历程中给予了支持和鼓励。Sandra、Kassy、Sammy、Ally、Ben、Joey、Mark 和 Peter，谢谢你们。

免责声明

感谢你阅读本书。虽然笔者试图对 Hadoop 生态系统的不同安全特性进行解释、文档化并给出建议，但这并不表明或暗示，使用这些特性中的任何一个将获得完全安全的集群。从安全角度来说，无论使用什么保护机制，任何信息系统都不是 100% 安全的。我们鼓励对 Hadoop 环境进行持续的安全检查，从而确保可能达到的最好的安全状态。对于可能由于使用本书中任何特性所造成的损坏，笔者、O'Reilly 公司及人民邮电出版社不对此负责。所有操作，风险自负。

电子书

扫描如下二维码，可购买本书电子版。



第 1 章

引言

早在 2003 年，谷歌就发表了一篇论文（http://research.google.com/archive/gfs.html），阐述了基于服务器集群的、面向海量数据存储的可扩展系统架构，该架构称为谷歌文件系统（**GFS**）。一年后，谷歌发表了另一篇论文（http://research.google.com/archive/mapreduce.html），阐述了一个名为 **MapReduce** 的编程模型。该模型利用 GFS 并行处理数据，实现了处理程序在数据存储处的本地化运行。几乎同时，Doug Cutting 与其他合作者也在搭建一个开源的网络爬虫，如今这个工具被称为 Apache Nutch（http://nutch.apache.org）。Nutch 的开发者们意识到，MapReduce 编程模型和 GFS 是很完美的分布式网络爬虫构建模块，于是他们着手实现基于这两个项目的新爬虫版本。这些组件后来从 Nutch 脱离，并组成了 Apache Hadoop 项目。围绕 Hadoop 可扩展架构形成的生态系统[1]允许用户存储和处理所有重要的事务数据，提供了解决问题的另一种方法。

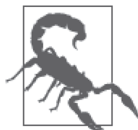
当所有这些崭新且令人激动的处理、存储数据的方法在 Hadoop 生态系统中被广泛使用时，使用单一的集中式集群管理这些 PB 级数据明显是有风险的。成百上千的服务器在同一个通用应用栈中互连，给如何保护这些重要资源提出了很多难题。其他书籍侧重于介绍如何编写 MapReduce 代码、如何设计最优的导入框架，以及如何在 Hadoop 生态系统上搭建复杂的低延迟处理系统，而本书专注于讨论在 Hadoop 的安全架构下，如何利用众多安全特性保护上述所有事务的安全性。

1.1 安全概览

开始讨论 Hadoop 专有内容之前，有必要先了解一些信息安全相关的关键理论和术语。信

注 1：Apache Hadoop 本身由 4 个子项目组成：HDFS、YARN、MapReduce 和 Hadoop Common。然而，Hadoop 生态系统、Hadoop 和其他在 Hadoop 基础上创建或集成的相关项目通常均被简称为 Hadoop。我们此处使用 Hadoop 项目和 Hadoop 生态系统以示区分。

息安全理论的核心是 **CIA** *模型*，即*机密性*（**confidentiality**）、*完整性*（**integrity**）和*可用性*（**availability**）。该模型的这 3 个组件是高层次概念，能够广泛应用于信息系统、计算平台以及（更适用于本书的）Hadoop。我们还将进一步了解*验证*、*授权*和*审计*这 3 个安全计算领域的要素，并将在全书深入探讨。



虽然 CIA 模型能够帮助建立一些信息安全原则，但需要指出的是，该模型并不是一个需要严格遵循的规范。Hadoop 平台的安全特性既有可能涉及多个 CIA 模型组件，也有可能连一个 CIA 组件也覆盖不到。

1.1.1　机密性

机密性这个安全原则强调，信息只应该被期望的接收者看到。例如，假设 Alice 通过邮件向 Bob 发送了一封信，那么只有当 Bob 是唯一能够阅读这封信的人时，才能认为这封信是机密的。尽管这看起来很简单，不过要保证机密性，还需要几个重要的安全概念。比如，如何使 Alice 知道她发送的信件被 Bob 本人读了？如果 Bob 本人读了信件，他如何确认自己所读的信是 Alice 本人发的？为了使 Alice 和 Bob 都参与机密信息的传递，他们需要持有能够将自己和其他人区别开来的唯一*身份标识*。此外，Alice 和 Bob 需要通过一个*身份验证*的过程证明自己的身份。身份和验证是 Hadoop 安全的核心组件，这部分将在第 5 章详细介绍。

机密性的另一个重要概念是*加密*。加密是这样一种机制，它将数学算法应用于信息片段，使加密后的输出内容对于非预期接收者不可读。只有期望的接收者能对加密消息进行*解密*，从而得到原始信息。数据加密有两种方式：*静态加密*和*动态加密*。静态加密是指，数据在非访问状态下就处于加密状态。例如，存储在硬盘上的加密文件就是静态加密的一个例子。动态加密也称在线加密，即在数据通过网络从一个地方发送到另外一个地方时进行加密。两种加密方式可以各自独立使用，也可以混合使用。第 9 章将介绍 Hadoop 的静态加密，而动态加密会在第 10 章和第 11 章进行介绍。

1.1.2　完整性

完整性是信息安全的重要组成部分。之前的例子中，当 Alice 发送信件给 Bob 时，如果 Charles 在 Alice 和 Bob 毫不知情的情况下，于传输过程中截获了信件，并对其内容进行了修改，这会怎样呢？ Bob 如何保证他接收到的信件和 Alice 发送的是完全一样的呢？这就是*数据完整性*的概念。数据完整性是信息安全的关键要素，尤其在包含高度敏感数据的行业中。设想一下，如果银行没有机制能够验证客户账户余额完整性，会发生什么？医院没有机制能够验证病人病历完整性，又会发生什么？政府没有验证情报秘密完整性的机制，可能会发生什么？即使机密性得到保证，但完整性得不到保证的数据也有被损坏的风险。完整性将在第 9 章和第 10 章讲解。

1.1.3　可用性

可用性与前两个要素不属于同一类规范。机密性和完整性同是众所周知的安全概念，而可

用性更多涉及的是操作准备。例如，如果 Alice 试图向 Bob 发信，但邮局关门了，那么信件就无法被送达。这样对于 Bob 来说，信件就是不可用的。数据或服务的可用性能够被普通的行为影响，如计划内的停工升级或应用安全补丁，同时也可能被诸如分布式拒绝服务攻击（DDoS）等安全事件影响。《Hadoop 技术详解》和《Hadoop 权威指南》介绍了如何对高可用性进行配置，本书将从安全视角介绍这些概念，参见第 3 章和第 10 章。

1.1.4　验证、授权和审计

验证、授权和审计（通常缩写为 AAA）指计算机安全领域的一个架构模式。在该模式中，使用服务的用户先要证明自己的身份，然后根据规则被授予权限，同时其操作被记录下来留待审计。与 AAA 紧密相联的一个概念是身份。身份是指系统如何将不同的实体、用户和服务区分开来，典型的代表是一个任意字符串，例如用户名或者诸如用户 ID（UID）的唯一号码。

深入了解 Hadoop 如何支持身份、认证、授权和审计之前，先考虑在一个很简单的情况下运用这些概念：在单台 Linux 服务器上使用 sudo 命令。我们来看看两个不同用户 Alice 和 Bob 进行的终端会话。在这台服务器上，Alice 的用户名是 *alice*，Bob 的用户名是 *bob*。如示例 1-1 所示，Alice 先登录。

示例 1-1　验证和授权

```
$ ssh alice@hadoop01
alice@hadoop01's password:
Last login: Wed Feb 12 15:26:55 2014 from 172.18.12.166
[alice@hadoop01 ~]$ sudo service sshd status
openssh-daemon (pid 1260) is running...
[alice@hadoop01 ~]$
```

示例 1-1 中，Alice 通过 SSH 登录，并且立即被提示输入她的口令。她的用户名 / 口令对被用于在 `/etc/passwd` 密码文件中验证其登录。完成这一步后，Alice 便被成功验证了用户名 `alice` 的身份。Alice 的下一步是使用 `sudo` 命令获得 `sshd` 服务，这需要超级用户的权限。这个命令执行成功，说明 Alice 被授权可以执行该命令。对于 `sudo` 命令，控制哪些用户能够被授予超级用户权限的规则存储在 `/etc/sudoers` 文件中，如示例 1-2 所示。

示例 1-2　/etc/sudoers

```
[root@hadoop01 ~]# cat /etc/sudoers
root ALL = (ALL) ALL
%wheel ALL = (ALL) NOPASSWD:ALL
[root@hadoop01 ~]#
```

示例 1-2 中，用户 root 被授予使用 `sudo` 执行任何命令的权限，用户组 `wheel` 被授予可无密码使用 `sudo` 执行任何命令的权限。这个例子中，系统的验证依赖于登录，而非一次新的验证要求。最后一个问题是，系统怎么知道 Alice 是 `wheel` 用户组的一员呢？在 UNIX 和 Linux 系统中，这通常由文件 `/etc/group` 控制。

现在，我们知道控制 Alice 的身份标识的有两个文件：`/etc/passwd`（示例 1-4）和 `/etc/group`（示例 1-3）。`/etc/passwd` 文件为她的用户名分配一个特有的 UID 和主目录，`/etc/`

group 文件更进一步定义了系统的组标识以及用户和组的从属关系。这些身份标识信息文件会在执行 sudo 命令时用到，同时用到的还有 /etc/sudoers 文件中定义的权限规则，它们用于验证 Alice 是否有权限执行请求的命令。

示例 1-3　/etc/group

```
[root@hadoop01 ~]# grep wheel /etc/group
wheel:x:10:alice
[root@hadoop01 ~]#
```

示例 1-4　/etc/passwd

```
[root@hadoop01 ~]# grepalice /etc/passwd
alice:x:1000:1000:Alice:/home/alice:/bin/bash
[root@hadoop01 ~]#
```

现在看看示例 1-5 中 Bob 的会话。

示例 1-5　授权失败

```
$ ssh bob@hadoop01
bob@hadoop01's password:
Last login: Wed Feb 12 15:30:54 2014 from 172.18.12.166
[bob@hadoop01 ~]$ sudo service sshd status

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for bob:
bob is not in the sudoers file.  This incident will be reported.
[bob@hadoop01 ~]$
```

本示例中，Bob 能够使用与 Alice 类似的方法进行身份验证，但当他试图使用 sudo 命令时，遇见了截然不同的状况。首先，Bob 执行命令时被系统再次要求输入口令；成功提交口令后，被系统拒绝以超级用户的权限执行该 service 命令。这是由于与 Alice 不同，Bob 不是 wheel 组的成员，因而不被授权执行 sudo 命令。

了解了身份、验证和授权，我们再来关注审计。对于用户与诸如 SSH 和 sudo 等安全服务的互动行为，Linux 会创建一个名为 /var/log/secure 的日志文件。该文件能够记录一个账户的某些行为，无论这些行为成功还是失败。若我们在 Alice 和 Bob 各自完成上述操作后查看该日志，能够看到示例 1-6 所示的输出（按照方便阅读的形式组织）。

示例 1-6　/var/log/secure

```
[root@hadoop01 ~]# tail -n 6 /var/log/secure
Feb 12 20:32:04 ip-172-25-3-79 sshd[3774]: Accepted password for
  alice from 172.18.12.166 port 65012 ssh2
Feb 12 20:32:04 ip-172-25-3-79 sshd[3774]: pam_unix(sshd:session):
  session opened for user alice by (uid=0)
Feb 12 20:32:33 ip-172-25-3-79 sudo:    alice : TTY=pts/0 ;
```

```
  PWD=/home/alice ; USER=root ; COMMAND=/sbin/service sshd status
Feb 12 20:33:15 ip-172-25-3-79 sshd[3799]: Accepted password for
  bob from 172.18.12.166 port 65017 ssh2
Feb 12 20:33:15 ip-172-25-3-79 sshd[3799]: pam_unix(sshd:session):
  session opened for user bob by (uid=0)
Feb 12 20:33:39 ip-172-25-3-79 sudo:     bob : user NOT in sudoers;
  TTY=pts/2 ; PWD=/home/bob ; USER=root ; COMMAND=/sbin/service sshd status
[root@hadoop01 ~]#
```

对于 Alice 和 Bob，他们通过 SSH 成功登录的事实均被记录下来，包括他们对 sudo 的使用记录。Alice 的例子中，系统记录了她成功使用 sudo 以 `root` 权限执行了 `/sbin/service sshd status` 命令。而 Bob 的例子中，系统则记录了他试图以 `root` 权限执行 `/sbin/service sshd status` 命令，但由于他不在 `/etc/sudoer` 用户组里，以致被系统拒绝执行。

该例子展示了身份、验证、授权和审计的概念如何运用在相对简单的单台 Linux 服务器上，并维持系统安全。这些概念将在第二部分的 Hadoop 相关章节中予以详细阐述。

1.2 Hadoop安全：简史

Hadoop 在存储和处理大量数据时效率很高，并且与其他平台相比更经济。Hadoop 项目最初的关注点是实际的技术实现，项目中许多代码涵盖的逻辑能够针对分布式系统中固有的复杂性进行处理，如故障处理与协同。由于这种较强的针对性，早期的 Hadoop 项目建立了一个安全立场：整个机器集群和所有访问它的用户都是*可信网络*的一部分。这实际上意味着，Hadoop 并没有强安全策略强制执行很多措施。

随着 Hadoop 项目的发展，显然，至少应当有一个机制能够对用户身份进行强有力的验证。Kerberos 被选作该项目的这种机制，它是一个完善的协议，如今广泛应用于 Microsoft Active Directory 等企业系统。在强认证策略之外，还要有强授权策略。强授权策略定义了单个用户被认证后能够做的事情。起初，授权策略是在单个组件上实现的，这意味着管理员需要在很多地方对授权控制进行定义。后来，在仍是孵化项目的 Apache Sentry 中，授权机制终于变得容易了一些。但就像我们将在第 6 章和第 7 章中看到的一样，目前尚未有一个能在整个生态系统中具有整体统筹性的授权机制。

Hadoop 安全的另一个仍在演变的方面是，通过加密和其他机密性机制进行的数据保护。在可信网络中，人们起初假定数据本来就是被保护的，不会被非授权用户访问，因为只有授权用户才能接入可信网络。之后，Hadoop 为节点间的数据传输添加了加密手段，对硬盘上的数据存储也进行了加密。在后续的讲解中，你能看到这些安全策略的进化是如何产生的，但为了快速入门，首先还是需要关注 Hadoop 的生态系统。

1.3 Hadoop组件和生态系统

本节，我们将纵览贯穿全书的 Hadoop 生态系统的各个组件，这有助于在开始讨论组件的安全之前先了解它们。已经精通下列组件的读者可以直接跳至下一节。除非另外声明，本书描述的安全特性适用的项目版本均如表 1-1 所示。

表1-1：项目版本[a]

项目	版本
Apache HDFS	2.3.0
Apache MapReduce (for MR1)	1.2.1
Apache YARN (for MR2)	2.3.0
Apache Hive	0.12.0
Cloudera Impala	2.0.0
Apache HBase	0.98.0
Apache Accumulo	1.6.0
Apache Solr	4.4.0
Apache Oozie	4.0.0
Cloudera Hue	3.5.0
Apache ZooKeeper	3.4.5
Apache Flume	1.5.0
Apache Sqoop	1.4.4
Apache Sentry (Incubating)	1.4.0-incubating

a 细心的读者会发现，上述列表有一些遗漏。我们特意没有在上表中提到 Apache Spark、Apache Ranger 和 Apache Knox。这些项目是 Hadoop 生态系统新增的组件，由于时间的限制，我们在这里忽略了它们。

1.3.1 Apache HDFS

Hadoop 分布式文件系统（**HDFS**）通常被认为是 Hadoop 生态系统中其余部分的基础。HDFS 是 Hadoop 的存储层，在系统存储容量和带宽总量线性增长时，能够对海量数据进行存储。HDFS 是能够跨越多个服务器的逻辑文件系统，每个服务器都可有多块硬盘。从安全角度理解这一点很重要，因为 HDFS 中的某个文件可以跨越 Hadoop 集群中的多个或全部服务器。这意味着，客户端与某文件的交互有可能需要与集群中的每一个节点进行通信。HDFS 的一个关键实现方法是，将文件分解成块（**block**），这使得前述的交互成为可能。每一个文件块被存储在集群中任何节点的任何物理驱动器上。此问题较为复杂，我们在这里不深入讨论，并忽略原理部分的细节。推荐有兴趣的读者阅读《Hadoop 权威指南》。关键的安全要点是：HDFS 中的所有文件都被分解为块，当使用 HDFS 的客户端读写这些文件时，需要通过网络与 Hadoop 集群中的所有服务器进行通信。

HDFS 是一种主 / 从（head/worker）架构，由两个基本组件构成：NameNode（主）和 DataNode（从）。附加的组件包括 JournalNode、HttpFS 和 NFS 网关。

NameNode

NameNode 负责跟踪 HDFS 中所有与文件相关的元数据，例如文件名、块位置、文件许可和副本。从安全角度需要知悉的是，那些读写文件的 HDFS 客户端总是与 NameNode 通信的。此外，NameNode 为整个 Hadoop 生态系统提供了一些重要的安全功能，这些内容将在之后进行阐述。

DataNode

DataNode 负责在 HDFS 中对数据块进行存储和读取。NameNode 会告诉正在读取某文件的 HDFS 客户端，集群中的哪个 DataNode 拥有客户端请求的数据。向 HDFS 写文件时，客户端向一个由 NameNode 指定的 DataNode 写入数据块。DataNode 由此建立一个通向其他 DataNode 的写管道，从而完成基于所需复制因子的写操作。

JournalNode

JournalNode 是 HDFS 中的一种特殊组件。HDFS 被配置为高可用性（**HA**）时，JournalNode 会接管 NameNode 写入 HDFS 元数据信息的职责。集群通常具有奇数个 JournalNode（3 个或 5 个），用以确保能满足对仲裁资源的要求（不能出现平局）。例如，如果要向 HDFS 写入一个新文件，该文件的元数据会被写入每一个 JournalNode。当多数 JournalNode 成功写入该信息，发生的改变（写入行为）会被认为是有效的。HDFS 客户端和 DataNode 不会直接与 JournalNode 互动。

HttpFS

HttpFS 是 HDFS 中负责向客户端提供与 NameNode 和 DataNode 连接代理的组件。该代理是一种 REST API，它允许客户端——无需与 HDFS 中的其他组件建立直连——通过代理使用 HDFS。HttpFS 在某些集群的架构中将会是关键的组件，稍后阐述。

NFS 网关

顾名思义，NFS 网关允许客户端像使用 NFS 文件系统那样使用 HDFS。NFS 网关实质上是一个帮助客户端和底层 HDFS 集群间进行 NFS 协议通信的守护进程。与 HttpFS 很相似，NFS 网关位于 HDFS 和客户端之间，因此能够为特定的集群架构提供安全边界。

KMS

Hadoop 关键管理服务，亦作 **KMS**，在 HDFS 的静态透明加密功能中扮演着重要角色。它的功能是作为 HDFS 客户端、NameNode 和关键服务之间的中间人，处理加密操作，如解密数据加密密钥和管理加密区密钥等。这部分内容将在第 9 章进行详细阐述。

1.3.2 Apache YARN

随着 Hadoop 的发展，MapReduce 即便拥有强大功能，也明显不再能够满足新的用例需求。许多问题不能利用 MapReduce 的编程模型轻易解决，人们需要一种能够适应新处理模型的更通用的框架。Apache YARN 就提供了这种能力。其他处理框架如 Impala、Spark 等，也将 YARN 用作它们的资源管理框架。虽说 YARN 提供的是更通用的资源管理框架，但 MapReduce 依然是运行在它之上的标准应用。这种运行在 YARN 上的 MapReduce 被认为是 v2 版，简称 MR2。YARN 的架构由以下几部分组成。

ResourceManager

ResourceManager 守护进程负责处理应用提交请求、分配 ApplicationMaster 任务，以及执行资源管理策略。

JobHistory Server

顾名思义，JobHistory Server 负责跟踪在 YARN 上运行的所有作业历史，包括运行时间、任务数量、写入 HDFS 的数据量等作业的度量。

NodeManager

NodeManager 守护进程负责执行 YARN 容器（**container**）中作业的各个任务，每个任务由虚拟内核（CPU 资源）和 RAM 资源组成，并可以按照各自需要申请一定数量的虚拟内核和内存，其最小值、最大值和增幅由 ResourceManager 定义。每个任务作为独立的进程在各自的 JVM 中运行。NodeManager 的一个重要作用是，启动名为 **ApplicationMaster** 的特殊任务，该任务负责管理指定应用的所有任务状态。YARN 将资源管理和任务管理区分开，这样每个作业都能执行各自的 ApplicationMaster，从而能够在大型集群中更好地扩展 YARN 应用。

1.3.3 Apache MapReduce

MapReduce 是与 HDFS 相对应的数据处理部分，提供最基本的数据批处理机制。在 YARN 上执行的 MapReduce 常被称为 MapReduce2 或 MR2，人们以此区分基于 YARN 的 MapReduce 和独立的 MapReduce 框架（后者被追加命名为 MR1）。MapReduce 作业由客户端提交给 MapReduce 框架，然后在 HDFS 中的一个数据子集（通常是一个特定目录）上执行。MapReduce 自身也是一个编程模型，允许多个服务器并行、独立地处理数据块。虽然 Hadoop 开发者需要了解 MapReduce 复杂的工作原理，但安全架构师多半不需要了解这些。后者需要知道的是，客户端提交作业到 MapReduce 框架，并且从那之后，由 MapReduce 框架负责客户代码在集群之间的分发和执行。客户端不需要与集群的任何节点进行交互以执行作业，而是作业本身会申请运行一定数量的任务（**task**）完成其工作。依据 MapReduce 框架的调度算法，每个任务会被分配到一个节点运行。



“MapReduce 框架在指定服务器上建立的各个任务以什么用户身份执行”，这取决于 Kerberos 是否开启。如果 Kerberos 没有开启，每个任务都会以 `mapred` 系统用户的身份执行。Kerberos 开启时，各个任务会以提交 MapReduce 作业的用户身份执行。但即便开启了 Kerberos，当另外一个组件或工具正在提交 MapReduce 作业时，可能无法立刻看出哪个用户在执行该作业下的 MapReduce 任务。涉及 Hive 身份模拟的相关细节讨论参见 5.2.4 节。

与 HDFS 类似，MapReduce 也是一个主 / 从式架构，包括两个主要部分。

JobTracker（主）

客户端向 MapReduce 框架提交作业时，与客户端进行通信的其实是 JobTracker。JobTracker 负责处理客户端的作业提交，并且决定作业应该如何按照设定好的方式运行，例如作业需要多少任务、哪个 TaskTracker 负责哪个任务等。另外，JobTracker 还负责处理安全和运行相关的功能，例如作业队列、调度池，以及用于验证的访问控制表等。最后，JobTracker 还负责处理作业的度量和其他相关信息，这些信息在作业执行的过程中会被各种 TaskTracker 发送给 JobTracker。JobTracker 包括资源管理和任务管理两部分，这两部分在 MR2 中被 ResourceManager 和 ApplicationMaster 分隔开。

TaskTracker（从）

TaskTracker 负责执行一个 MapReduce 作业中的给定任务。TaskTracker 从 JobTracker 接收要运行的任务，并为每个运行的任务生成独立的 JVM 进程。TaskTracker 既执行

map 任务，也执行 reduce 任务，具体可以并发执行多少个任务取决于 MapReduce 的配置。从安全角度看，其要点是，JobTracker 决定运行什么任务以及任务运行在哪些 TaskTracker 上。通常的作业执行过程中，客户端既不能控制任务如何分配，也不与 TaskTracker 进行任何通信。

关于 MapReduce 的一个要点是，其他的 Hadoop 生态系统组件都是建立在 MapReduce 之上的框架和库。这意味着，虽然 MapReduce 负责实际的数据处理，但这些框架和库将 MapReduce 的作业执行从客户端中抽象出来。Hive、Pig、Sqoop 组件都这样使用 MapReduce。



Hadoop 中，用户审计的一个重要部分就是理解 MapReduce 作业是如何提交的，具体细节参 5.2.3 节“块访问令牌”部分。从安全角度看，同样是使用 MapReduce，用户提交自己的 Java MapReduce 代码与使用 Sqoop 从 RDBMS 导入数据或在 Hive 中执行 SQL 查询相比，是完全不同的活动。

1.3.4 Apache Hive

Apache Hive 项目最早由 Facebook 发起。Facebook 看到了 MapReduce 的实用性，但也发现，分析人员缺乏 Java 编程技能，导致 MapReduce 框架的适用性大打折扣。由于大部分 Facebook 分析人员具备 SQL 技能，因此 Facebook 发起了 Hive 项目，使用 MapReduce 作为执行引擎的 SQL 抽象层。Hive 架构由以下几部分组成。

Metastore 数据库

Metastore 数据库是一个包含所有 Hive 元数据（例如库、表、列、数据类型等信息）的关系型数据库。这种信息在访问 HDFS 时将数据结构化，也被称为“读模式”（**schema on read**）。

Metastore Server

Hive Metastore Server 是处于 Hive 客户端和 Metastore 数据库之间的一个守护进程，它提供了一个安全层，防止客户端拥有 HiveMetastore 的数据库凭证。

HiveServer2

HiveServer2 是客户端使用 Hive 的主入口。HiveServer2 兼容 JDBC 和 ODBC 客户端，因此被很多客户端工具和其他第三方应用使用。

HCatalog

Hcatalog 是一系列帮助非 Hive 框架访问 Hive 元数据的库。例如 Pig 用户可以用 Hcatalog 读取与 HDFS 中给定目录相关的模式信息。WebHCat Server 是一个向客户端提供 REST 接口的守护进程，使客户端能够访问 HCatalog 的 API。

如果想全面了解 Hive，可以阅读《Hive 编程指南》。

1.3.5 Cloudera Impala

Cloudera Impala 是一个大规模并行处理（Massive Parallel Processing，MPP）框架，专门为

了分析型 SQL（Analytic SQL）而建立。Impala 从 HDFS 读取数据，并利用 Hive Metastore 解析数据结构和数据格式。Impala 架构由以下几部分组成。

Impala 守护进程（impalad）

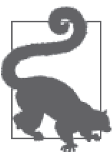
Impala 守护进程负责数据处理中的所有繁重任务，这些服务与 HDFS DataNode 相配合，以优化本地读取。

StateStore

StateStore 服务保存所有正在运行的 Impala 守护进程的状态信息。它监视 Impala 守护进程的状态是开启还是关闭，并向所有服务广播该状态。StateStore 并不是 Impala 架构中必需的组件，但它可以在一个或多个服务宕掉时提供更快的容错能力。

Catalog Server

Catalog Server 是从 Impala 到 Hive Metastore 的入口。该进程负责从 Hive Metastore 获取元数据，以及对 Impala 客户端修改过的元数据进行同步。有一个独立的 Catalog Server 既能够降低 Hive Metastore Server 的负载，也能够为 Impala 额外提供速度优化。



接触 Hadoop 生态系统的新用户经常会问，Hive 和 Impala 都提供对 HDFS 数据的 SQL 访问能力，那它们的区别是什么？ Hive 让熟悉 SQL 的人在不需要对 MapReduce 进行任何了解的前提下访问 HDFS 数据，它的设计目标是对 MapReduce 的内部结构进行抽象，从而使得访问 HDFS 数据更加容易，广泛应用于批处理和 ETL（Extract-Transform-Load，即数据的抽取、转换、加载）工作；而 Impala 则是自底向上设计的快速分析型处理引擎，支持即席查询和商业智能（business intelligence，BI）工具。Hive 和 Impala 都很实用，应当视为互补组件。

如果想全面了解 Impala，请参考 *Getting Started with Impala*。

1.3.6 Apache Sentry

Sentry 组件（孵化中）为其他生态系统组件（如 Hive、Impala 等）提供细粒度角色访问控制（role-based access control，RBAC）。虽然各个组件可能会有自己的验证机制，但 Sentry 在各个组件之间提供了一个支持强制集中策略的统一验证机制。Sentry 是 Hadoop 安全的一个重要组件，我们将用整整一章介绍它（第 7 章）。Sentry 由以下几部分组成。

Sentry Server

Sentry server 是一个守护进程，方便查找其他 Hadoop 生态系统组件产生的策略。Sentry 的客户端组件会根据 Sentry 置入的策略继承验证决策。

策略库

Sentry 策略库存储所有验证策略。Sentry Server 根据策略库决定一个用户的特定操作是否被允许。具体来说就是，Sentry Server 会为用户寻找一个匹配的策略，以授予其对资源的访问权限。早期版本的 Sentry 中，策略库仅是一个包含所有策略的文本文件。Sentry 和策略库的发展会在第 7 章详细讨论。

1.3.7　Apache HBase

Apache HBase 是分布式键 / 值存储，由谷歌的 BigTable 论文“BigTable: A Distributed Storage System for Structured Data”启发而来。HBase 通常使用 HDFS 作为数据的底层存储层，结合本书要讨论的内容，我们假定书中提到的 HBase 均为这种情况。HBase 表被划分到不同区域，这些区域根据行键（**row key**，键值的索引部分）进行分隔。行 ID 是顺序排列的，所以一个指定的区域具有一系列顺序的行键。区域由 **RegionServer** 存放，客户端使用键值从 RegionServer 请求数据。键值包括几个部分：行键、列族（**column family**）、列限定符（**column qualifier**）和时间戳（**timestamp**）。这些部分组合起来能够唯一地确定存储在表中的一个值。

客户端访问 HBase 时，首先通过搜索 `hbase:meta` 表查找负责存放特定范围行键的 RegionServer。找到正确的 RegionServer 后，客户端会直接向该 RegionServer 发起读写请求，而不是通过 Master 发起。客户端会缓存 RegionServer 的区域映射，以避免重复的查询过程。存放 `hbase:meta` 表的服务器位置可以在 ZooKeeper 中查到。HBase 包括以下几部分。

Master

正如之前提到的，HBase Master 守护进程负责管理哪个区域由哪个 RegionServer 存放。如果某个 RegionServer 宕掉，HBaseMaster 也要负责将它存放的区域重新分配给其他 RegionServer。多个 HBaseMaster 可以并行运行，在任意时刻，它们会通过 ZooKeeper 选择一个 HBaseMaster 保持活跃。

RegionServer

RegionServer 负责为指定的 HBase 表提供区域。区域是一系列特定范围的键值，这些键值既可以通过 HBase 控制台人工确定，也可以由 HBase 根据曾经存入表的键值自动确定。HBase 的一个目标是平均分配键值空间，使每个 RegionServer 具有同等的提供数据的责任。每个 RegionServer 通常存放着多个区域。

REST Server

HBase REST Server 提供执行 HBase 指令的 REST API。和 Hadoop 生态系统中的许多其他项目一样，默认的 HBase API 是用 Java API 提供的。REST API 通常被用作一种独立于编程语言的接口，使客户端能够使用任何他们想用的编程语言。

Thrift Server

除 REST Server 之外，HBase 还有一个 Thrift Server，它为客户端提供了另外一种可用的 API 接口。

要想了解更多 HBase 架构和最适用的用例，推荐阅读《HBase 权威指南》。

1.3.8　Apache Accumulo

Apache Accumulo（http://accumulo.apache.org）是一个排序分布式的键 / 值存储，一个健壮的、可扩展的、高性能的存储和检索系统。与 HBase 类似，Accumulo 起初也是基于 Google BigTable 设计的，但它建立在 Apache Hadoop 生态系统（尤其是 HDFS、ZooKeeper 和 Apache Thrift）之上。Accumulo 使用的数据模型和 HBase 大体一致，每个 Accumulo

表被分成一个或多个块，每个块包含基本相同数量的记录，这些记录根据行 ID 分配。每个记录也有一个由多部分组成的列键，包括列族、列限定符和一个可见性标签（visibility label）。可见性标签是 Accumulo 与原始 BigTable 设计最大的不同点之一，它增加了实现单元级安全的能力（我们将会在第 6 章详细讨论）。最后，每个记录也包含一个时间戳，使用户可以存储多个版本的记录。如果没有时间戳，这些不同版本的记录的键值将相同。综上，行 ID、列和时间戳组成了确定一个特定记录的键值。

块的分配以分割行 ID 集合的方式进行，分割点在数据被插入表时自动计算。每个块由一个负责在该块中提供数据读写服务的 TabletServer 存放，每个 TabletServer 可以存放来自一个或多个表的多个块，所以块是系统中的分配单元。

客户端第一次访问 Accumulo 时，会查找存放 `accumulo.root` 表的 TabletServer 位置，`accumulo.root` 表保存着 `accumulo.meta` 如何被分割到块的信息。客户端会直接与存放 `accumulo.root` 的 TabletServer 通信，然后再与存放 `accumulo.meta` 表数据块的 TabletServer 通信。由于这些表中（尤其是 `accumulo.root` 中）数据的改变频率与其他数据相比相对较少，所以客户端会保存一个从这些表读取的数据块位置的缓存，以避免读写管道中的瓶颈。一旦客户端知道了它正在读 / 写的行 ID 所属的数据块位置，它就会直接与对应的 TabletServer 通信。客户端在任何时候都不会与 Master 通信，这一点在很大程度上保证了可扩展性。总体来说，Accumulo 包含以下几部分。

Master

Accumulo Master 负责协调分配数据块到 TabletServer，保证每个块被存放在唯一的 TabletServer 中，并对 TableServer 失效等事件进行响应。它还负责表的管理性更改以及协调启动、关闭和写前日志的恢复。多个 Master 可以同时运行，它们会选出一个领导者，从而使每个时刻只有一个 Master 处于活跃状态。

TabletServer

TabletServer 负责处理对 Accumulo 集群中数据块子集的所有读写请求。就写操作而言，它负责周期性地将记录写入写前日志，并将内存中的记录写入磁盘。恢复过程中，TabletServer 将写前日志中的记录回放到要恢复的数据块。

GarbageCollector

GarbageCollector 周期性地删除 Accumulo 进程不再需要的文件。多个 GarbageCollectors 可以同时运行，它们会选择一个领导者，从而使每个时刻只有一个 GarbageCollector 处于活跃状态。

Tracer

Tracer 监控集群中使用 Accumulo 分布式定时 API 的其他部分，并将这些数据写入一个 Accumulo 表，以备后用。多个 Tracer 可以同时运行，它们会平均分配负载。

Monitor

Monitor 是一个用于监控 Accumulo 集群状态的 Web 应用，展示了记录数量、缓存命中率 / 未命中率等关键度量值，以及扫描率等数据表信息。Monitor 还充当了日志转发节点的角色，使用户可以通过单一接口诊断错误和警告信息。

1.3.9 Apache Solr

Apache Solr 项目又称 **SolrCloud**，能够对分散于不同物理服务器的文档（作为更大数据集合的一部分）进行搜索和检索。搜索是大数据的经典用例之一，也是任何人访问互联网都会用到的最常见的功能。Solr 建立在 Apache Lucene 项目之上，由 Lucene 实际负责大部分的检索和搜索工作。Solr 提供逐级导航（faceted navigation）、高速缓存、命中字高亮显示、管理接口等企业级功能，以对这些能力进行扩展。

Solr 是一个单独的组件，即 Server。一个部署环境中可以有多个 Solr Server，并通过 SolrCloud 提供的分片机制线性扩展。SolrCloud 还提供了自我复制功能，以应对分布式环境中的故障。

1.3.10 Apache Oozie

Apache Oozie 是一个 Hadoop 工作流和业务流管理系统。它可以建立包含多个动作的工作流，每个动作可以使用 Hadoop 生态系统中的不同组件。例如，一个 Oozie 工作流可以首先执行一个 Sqoop 导入动作，将数据移入 HDFS；然后执行一个 Pig 脚本，对数据进行转换；之后再用一个 Hive 脚本建立元数据结构。Oozie 支持多个复杂的工作流，例如使用分叉（fork）和连接（join）动作可以并行执行多个步骤，或者执行依赖于多个步骤完成之后才能继续的其他步骤。Oozie 工作流可以根据不同输入条件，按照周期性的计划运行，例如在特定时间运行，或者等待 HDFS 中某个路径存在后运行。

Oozie 只有一个服务组件，该服务负责处理客户端的工作流提交、管理工作流的执行以及状态报告。

1.3.11 Apache ZooKeeper

Apache ZooKeeper 分布式协同服务允许分布式系统同步存储和读取小量数据，常用于存储通用配置信息。此外，ZooKeeper 还经常用于同步 Hadoop 系统中的高可用性服务，例如元数据节点 HA 和资源管理 HA。

ZooKeeper 本身也是一个分布式系统，它通过奇数个 ZooKeeper **Ensemble** 服务器的仲裁，确认下发的事务。ZooKeeper 只有一个组件，即 ZooKeeper Server。

1.3.12 Apache Flume

Apache Flume 是一个基于事件的数据导入工具，主要用于将数据导入 Hadoop，也可完全独立使用。Flume 的意思是槽，顾名思义，其作用是将事件日志导入 HDFS。Flume 框架主要由三部分组成：数据源（source）、数据阱（sink）和通道（channel）。

Flume 的数据源定义了应当如何从上游提供者读取数据。读取流程可能包括 syslog 服务器、JMS 队列，甚至轮询某个 Linux 目录等。Flume 数据阱定义了数据应当如何被写入下游。通用的数据阱包括一个 HDSF 阱节点和一个 HBase 阱节点。最后，Flume 通道定义了数据在源和阱之间是如何被存储的。最基本的两种通道是内存通道和文件通道。内存通道通过牺牲一定的可靠性实现高速传输，文件通道则通过牺牲一定的传输速度提供较高的可靠性。

Flume 中只有一个组件，即 Flume **Agent**。Agent 包含数据源、数据阱和通道的代码。Flume 架构的一个重要特性是：Flume Agent 之间能够互联，即某个 Agent 的数据阱能够与另一个 Agent 的数据源连通。这种情况下的一个通用接口是 Avro 源和阱。Flume 数据导入和安全将在第 10 章介绍，也可翻阅 *Using Flume*（http://shop.oreilly.com/product/0636920030348.do）。

1.3.13 Apache Sqoop

Apache Sqoop 提供了在传统 RDBMS 以及其他数据源（如 FTP 服务器）中批量导入 / 导出数据的功能。Sqoop 自身通过提交 map-only 的 MapReduce 任务，与 RDBMS 进行并行交互。Sqoop 既可作为初始化 Hadoop 集群数据的一种简单机制，也可以是进行常规数据导入 / 导出工作的一种工具。目前有两种不同版本的 Sqoop：Sqoop1 和 Sqoop2。本书重点讲解 Sqoop1。Sqoop2 在本书写作之时尚未完善，并且缺乏一些基本的安全特性，如 Kerberos 身份验证。

Sqoop1 是由使用 sqoop 程序的命令行衍生的一系列客户端库。这些客户端库负责实际的 MapReduce 作业提交，即将作业提交到合适的框架（例如传统的 MapReduce 或者基于 YARN 的 MapReduce2）。Sqoop 的详细讨论参见第 10 章，也可翻阅 *Apache Sqoop Cookbook*（http://shop.oreilly.com/product/0636920029519.do）。

1.3.14 Cloudera Hue

Cloudera Hue 是一个 Web 应用，将许多 Hadoop 生态系统组件友好地暴露给用户。Hue 允许用户无需熟悉 Linux 或者各种命令行接口，就可轻松访问 Hadoop 集群。Hue 拥有一些不同的安全控制策略，这些将在第 12 章讲到。Hue 由如下组件组成。

Hue Server

Hue Server 是 Hue 的主要组件，它实质上是一个向用户提供网页内容的 Web 服务。用户第一次登录时需要进行验证，之后，终端用户执行的动作实质上是由 Hue 代理用户完成的。这就是第 5 章将介绍的*身份模拟*（**impersonation**）。

Kerberos Ticket Renewer（**Kerberos** *票据更新器*）

顾名思义，该组件负责周期性地更新 Kerberos 的*票据授予票据*（**ticket-granting ticket**，**TGT**），在 Hadoop 集群启用 Kerboros 的前提下，该票据被 Hue 用于与 Hadoop 集群进行交互（第 4 章对 Kerberos 有详细阐述）。

1.4 小结

本章介绍了一些常见的安全术语，这些安全术语是本书其他内容的基础。阅读本章的一个关键收获是，读者能够认识到，Hadoop 安全并非那么难以理解。传统可靠的安全准则（如 CIA 和 AAA）在 Hadoop 环境中同样适用，这些会在后面的章节中详细讨论。最后，我们回顾了很多 Hadoop 生态系统项目（及其组件），并逐个了解它们的作用，并大致理解安全是如何应用的。

下一章我们会讨论保护分布式系统的安全。你会发现，很多 Hadoop 上的安全威胁和缓减方法也同样适用于分布式系统。

第 一 部 分

安全架构



第 2 章

保护分布式系统

第 1 章介绍了安全计算的一些关键原理，本章将进一步了解思考分布式系统安全时面临的一些有趣挑战。接下来你将看到，分布式大大增加了系统的潜在威胁，从而也增加了为帮助减轻这些威胁而必需的安全评估的复杂度。我们将用一个真实的案例说明安全需求如何随着系统的分布式程度而增加。

以银行为例。很多年前，对普通人而言，银行业务就是开车去当地银行，找个出纳员，然后亲自进行交易。而银行的安全措施可能包括检查客户的身份和账号，以及验证用户要求的操作可以执行，比如在取现时确保账户内有足够的钱。

多年来，银行越来越大。小镇上的银行变成了一个更大银行的支行，因此，你不仅可以在附近的这个银行支行办理业务，也可以在其他地方的该银行支行办理。这时，保护资产所必需的安全措施也有所增加，因为需要保护的不再是一个地点。同时，更多的银行出纳员也需要进行妥善的培训。

更进一步，银行终于使用上了 ATM 机，客户因此不需要去支行就能取现。你可能会想到，与以前银行业务在人与人之间进行交互的情况相比，这时需要更多安全措施保护银行。随后，银行之间开始相互联网，从而允许一家银行的客户使用另一家银行的 ATM 机。这时，银行需要相互之间建立安全控制措施，以确保这种互联不会造成安全损失。最后，互联网运动引入了通过网站甚至移动设备进行网上银行业务的能力，这又大大增加了银行的潜在威胁，以及对安全控制措施的需求。

如你所见，刚开始，保护小镇上的小银行只是一个简单的安全任务。随着银行在数十年来变得越来越分散和互联，难度增加了好几个数量级。虽然这个例子看起来很浅显，但它表达了如何为分布到数十、数百甚至数千台机器上的系统设计安全框架的问题。这是个不小的任务，为了使这个任务不那么可怕，我们可以对它进行拆解。首先，从理解威胁开始。

2.1 威胁种类

要想为分布式系统设计健壮的安全架构，一个关键的要素是理解可能出现的威胁，并能够对威胁进行分类，从而更好地理解要降低这些威胁需要何种安全机制。本节将回顾一些需要关注的、常见的威胁类别。

2.1.1 非授权访问 / 伪装

非授权访问是最常见的威胁类别之一。这种情况出现在某个人应当被拒绝访问时，反而成功进入系统。非授权访问的一种常见方法是*伪装*攻击。伪装的概念是，一个非法用户冒充为一个合法用户，从而获取访问权。你可能会问："非法用户是如何冒充为合法用户的？"最可能的答案是，攻击者获取了一个合法用户的用户名和口令。

伪装攻击自从互联网时代开始就特别突出，尤其是针对分布式系统的伪装攻击。攻击者有很多种方法获取合法的用户名和口令，比如尝试常见的单词和词组，或者了解与合法用户相关的、可能会被用作密码的单词。比如攻击者想获取一个社交媒体网站的登录凭证，那么他可能会从用户的公开帖子中收集关键字，并组成可供尝试的密码列表。举个例子，如果攻击者关注身在纽约、将"棒球"列为兴趣爱好的用户，那么他可能会尝试密码 yankees（yankees 是纽约扬基队的队名）。

如果非法用户成功实施了伪装攻击，安全管理员如何才能得知呢？毕竟攻击者登录时使用的是合法用户的凭证，从分布式系统的角度看，这不是正常行为吗？不见得。通常情况下，可以查看审计日志中的尝试登录行为以发现伪装攻击。如果攻击者使用可能的密码列表对某个用户账号进行尝试，那么不成功的尝试行为应当出现在审计日志文件中。如果看到有针对某个用户的大量失败的尝试登录行为，那么通常可归因于攻击。一个合法用户有可能输错或者忘记密码，从而导致少量失败的尝试登录事件，但比如 20 次连续的登录失败就应该是异常的了。

另外一种发现伪装攻击的常见途径是，从网络层面看，登录尝试的来源是哪里。按照 IP 地址分析登录尝试行为是发现是否有人在尝试伪装攻击的好方法。尝试登录的客户端 IP 地址是否与期望相符，比如是来自公司 IP 地址的一个已知子网，还是来自世界另一端的某个国家？还有，登录尝试行为发生在什么时间？ Alice 是在凌晨 3 点还是在正常的工作时间尝试登录系统的？

非授权访问的另外一种形式是攻击者利用系统的漏洞，从而不需要提供合法凭证就能进入系统。我们将在 2.3 节讨论漏洞。

2.1.2 内在威胁

*内在威胁*可以说是最具破坏性的威胁类型。顾名思义，攻击者来自业务内部，并且是一个正常用户。内在威胁可以包括员工、顾问、承包商。内在威胁之所以可怕是因为，攻击者已经具有系统的内部访问权限。攻击者可以使用合法凭证登录并获得系统授权，以执行特定功能，通过途中的许多安全检查（因为攻击者*被认为*应当被授予访问权限）。这种威胁可以导致对系统的肆无忌惮的攻击，或者诸如攻击者利用自己的权限把敏感数据泄漏给非

授权用户等更为巧妙的攻击。

本书中，你将会找到确保用户只访问他们所需的数据和服务的安全特性。对付内在威胁需要有效的审计实践（参见第 8 章）。除了能够帮助对抗内在威胁的技术工具之外，还需要建立业务策略强制实施合适的审计，对事件响应的流程也需要明确。虽然此处并未涉及如何建立这种策略的最佳实践，但这些策略对本章描述的所有威胁类型而言都是必要的。

2.1.3 拒绝服务

拒绝服务（**Denial of Service**，**DoS**）是指，一个服务对于一个或多个用户而言不可用。“*服务*”在这里是一个概括性词汇，包括数据访问、处理能力以及相关系统的可用性。拒绝服务的发生可以来自各种各样的攻击方式。在互联网时代，一种常见的攻击方式是，简单地使用大量网络流量压垮系统。这需要使用很多机器并行进行，因此使得这种攻击变为*分布式拒绝服务*（**Distributed Denial of Service**，**DDoS**）。当系统被过多需要处理的请求轰炸，就会开始在某些方面产生故障，小到丢弃其他合法请求，大到系统完全失效。

虽然分布式系统通常受益于其具有的某些容错能力，但 DoS 攻击仍然是可能的。例如，如果一个分布式系统包含 50 台服务器，攻击者可能很难扰乱全部 50 台设备的服务。但如果该分布式系统仅仅部署在少量网络设备后面，例如 1 台网络防火墙和 1 台接入交换机呢？攻击者可以将分布式系统的网关——而不是分布式系统本身——作为目标，通过这种方法达到自己的目的。这一点很重要，我们将在第 3 章——讲述建立集群周边的网络边界——对其进行讨论。

2.1.4 数据威胁

数据是分布式系统中最重要的部分。没有了数据，一个分布式系统就只是数据中心里一堆“嗡嗡”响的空闲服务器，只能徒增电力和冷却费用。正因为数据如此重要，它也是安全攻击的重点。数据威胁在一个分布式系统中出现在多个地方。首先，数据必须以安全的方式存储，以防止非授权的查看、篡改或删除。其次，数据*在传输过程中*也必须得到保护，因为分布式系统毕竟是分布式的。通过网络传递数据可能会被 DoS 攻击等破坏性的攻击威胁，也可能会被一些更为被动的攻击方式威胁，例如攻击者在通信方不知道的情况下抓取网络通信数据。第 1 章讨论了 CIA 模型及其组件，CIA 模型在根本上就是关于降低数据威胁的。

2.2 威胁和风险评估

你可能不是第一次听到上一节中对威胁类别的叙述，在理解这些威胁类别的基础上，对实际的分布式系统进行风险评估是很重要的。例如，拒绝服务攻击更有可能发生在直接连接到互联网的系统上，而对于那些不能从外网访问的系统，如公司内网中的系统，这种攻击发生的风险就低得多。注意，是风险低而不是没有风险，这是个很重要的区别。

评估对分布式系统的威胁需要进一步了解两个要素：用户和环境。理解这两个要素后，评估风险就更容易了。

2.2.1 用户评估

了解分布式系统会暴露给什么用户是很重要的。这明显包括访问系统的用户和直接与系统接口交互的用户，也包括可能出现在别处但不会直接访问系统的用户。理解这种环境下的用户有助于更好地进行风险评估。首先根据分布式系统（如 Hadoop）的用户工作对其进行分类。用户是做什么的？他们是业务情报分析员吗？或者是开发者？还是风险分析员？还是安全审计员？抑或是数据质量分析员？

一旦用户根据业务功能被分成不同的组，就可以开始识别不同组用户使用分布式系统的访问模式和工具。例如，如果分布式系统的用户都是开发者，那么可以假定他们需要系统中节点的控制台（shell）访问、调试作业的日志文件和开发工具。另一方面，业务情报分析人员可能不需要以上任何一种，而需要一套代替用户与分布式系统交互的分析工具。

还有间接访问系统的用户。这些用户不需要系统数据或处理资源的访问权限，但他们仍会与系统进行交互。这种交互作为某些功能的一部分，比如一些系统维护、健康监控、用户审计的支持功能。这种类型的用户需要被考虑到总体的安全模型中。

2.2.2 环境评估

为了对分布式系统进行风险评估，还需要了解系统所在环境。这通常意味着要对本系统与其他逻辑系统相关的、物理环境相关的操作环境都进行评估。第 3 章将介绍 Hadoop 专有的相关内容。

如前所述，环境评估的关键准则之一是，判断分布式系统能否通过互联网访问。如果能，则说明会有大量威胁可能发生，如 DoS 攻击、漏洞利用和病毒。连接至互联网的分布式系统需要长期监测，就像应用软件需要定期打补丁、安全软件需要定期升级一样。

对环境评估的另一准则是，要了解组成分布式系统的服务器的物理位置。它们位于自己公司的数据中心吗？还是位于第三方管理的数据中心？抑或是部署在公有云设施上？了解这些问题的答案会有助于开始建立安全评估的框架。例如，若分布式系统搭建在一个公有云上，那么这些威胁会立刻显现：基础架构不由自己的公司拥有，因此用户不清楚谁对这些机器拥有直接访问权，这使得主机服务提供商也被囊括进内部威胁之中。同样，公有云的使用回避了用户如何连接至分布式系统，以及数据如何流入 / 流出系统的问题。一个开放网络到一个共享的公有云之间的通信受到的威胁级别要比公司内部数据中心之间通信的威胁级别高得多。

这样说的重点不是要让大家认为，公有云不好而公司的数据中心好；而是要告诉读者，对于分布式系统已知威胁的级别会由于其存在形式的不同而不同。若不考虑环境因素，那么保护分布式系统的关键在于从多种层面降低风险，这些内容将会在下一节讨论。

2.3 漏洞

虽然漏洞是一个独立话题，但也与威胁和风险相关。分布式系统中，漏洞以多种不同形态存在。一个常见的存在漏洞的地方是软件本身，所有软件都有漏洞。这听起来像是个武断的结论，但事实是，没有软件是 100% 安全的。

那么，到底什么是软件漏洞呢？简言之，漏洞就是一段代码，这段代码容易受到未被考虑的错误或者无效条件的影响。下面看一个简单的实例。一个软件含有一个输入密码的界面，该界面允许用户修改密码（假设软件的内在逻辑允许至多 16 个字符长度的密码）。如果新密码的输入框长度被错误地限制为至多 8 个字符，导致用户选择的新密码被错误截断，那么会发生什么呢？这可能导致用户实际上设定的密码比他们认为的要短，或者导致更糟的情况：复杂度较低的密码更容易被攻击者猜出。

当然，软件漏洞不是分布式系统容易遭受的唯一漏洞，其他漏洞还与分布式系统依赖的网络架构设施有关。例如，多年前有个漏洞，允许攻击者向网络的广播地址发送 ping 命令，这导致网络中的每一台主机都会回复 ping 命令。攻击者构造 ping 的请求，将发起请求的源 IP 设定为网络中其试图攻击的某个主机 IP。这样做的结果就是，目标主机被网络中的通信淹没，从而导致故障。这种攻击手段通常称为“*死亡之* **ping**”（**ping of death**）。该风险现在已经缓解，但重点是，虽然该漏洞与网络中机器的软件没有关系，但在网络硬件供应商修复该漏洞之前，攻击者依然能够利用这个漏洞破坏网络中的特定主机。

软件补丁通常用于修复被发现的漏洞，因此，定期按计划更新补丁包是分布式系统软件栈中每个管理员的标准操作规程中不可缺少的部分。正如“死亡之 ping”的例子展示的那样，补丁的范围也应包括交换机、路由器、其他网络设备、硬盘控制器以及服务器 BIOS 的固件。

2.4 深度防御

安全管理员面临的众多挑战之一是，如何缓解本章提到的所有威胁和漏洞。研究多种威胁时，人们很容易发现，没有一个“放之四海而皆准”的方法能够有效消除这些威胁。为了防御这些威胁，提供一个可接受的安全等级，需要部署很多安全控制策略——并且它们必须协同工作。这种同时部署多种安全控制策略和保护措施的方法就是*深度防御*。

回顾历史，深度防御并未被常常遵守。“安全”通常指*边界安全*，此概念中，安全控制策略仅仅存在于受保护目标的外部或边界。一个经典的例证便是想象一个堡垒，其周边围有很厚的高墙。一般的思维定势会认为，只要高墙不倒，堡垒就是安全的。如果高墙被攻陷，堡垒里的人就危险了。如今，情况好了很多。

现在，深度防御存在于我们的日常生活。以去杂货店为例。杂货店有一扇上了锁的门，只有日间正常营业时间才会打开。同时，店里也有警报系统，入侵者在非营业时间非法闯入时会触发。在正常的营业时间，购物者的行为被遍布商店的安全摄像头监控着。最后，商店雇员也受到专门培训，以注意那些形迹可疑的客户。

所有这些安全措施的部署都是为了针对多种不同威胁，例如入室盗窃、伪装成顾客的扒手，以及抢劫等，以保护商店。如果杂货店仅仅依赖于“城堡高墙”的策略加强门和锁，将会对大部分威胁束手无策。深度防御很重要，因为任何单一的安全策略均无法缓解商店面临的所有风险。对于分布式系统也一样，在很多地方能够部署独立的安全策略。例如在边界配置网络防火墙，严格限制对数据的访问，或者限制对服务器的访问等，但同时使用所有这些策略才能降低攻击的成功率。

2.5 小结

本章通过分析威胁种类和漏洞分解了分布式系统的安全体系，并且展示了使用深度防御的安全架构能够将安全风险减少到最低。我们还讨论了内部威胁，以及为什么不能在设计安全架构时忽视它。

第 3 章将从建立一个健壮的系统架构开始，重点探讨 Hadoop 的保护措施。

第 3 章

系统架构

在第 2 章我们看到，随着单个独立的系统变为完全分布式的网络系统，其安全形势也发生了改变。我们能清楚地看到，保护一个 Hadoop 集群中成百上千的服务器有多么困难。本章将把集群分解为若干组件，这些组件可作为整体安全策略的一部分进行单独保护，从而深入探讨这个艰巨的任务。Hadoop 集群大致可分为两个主要部分：网络和主机。在此之前，先看看 Hadoop 集群的运行环境。

3.1 运行环境

在 Hadoop 早期，*集群*可能意味着一堆原本被用于其他用途，而现在被用来尝试新技术的机器，甚至可能是用一些陈旧的桌面级机器加上一些接入交换机组成的。多年来，事情发生了巨大的改变。在房间角落堆一些机器的日子已经被“Hadoop 集群是企业中的头等公民”的新观念所取代。Hadoop 集群在物理上和逻辑上融入企业的地方被称为“*运行环境*”。

影响 Hadoop 运行环境选择的因素有很多，已经超出了本书的范畴，我们将重点关注现在使用的典型操作环境。由于服务器和网络硬件的快速发展（感谢摩尔定律），Hadoop 可以在一些不同环境下运行。

内部（**In-house**）

该 Hadoop 环境包含企业自己拥有和运营的一系列物理机器，并且在企业自己掌控的数据中心里运行。

管理（**Managed**）

这种 Hadoop 环境是内部环境的一种演变，也由物理机器组成，但企业并不拥有和运营这些机器。它们是从另一个单独的企业租来的，该企业负责服务器的所有供给和维护工作。服务器在它们自己的数据中心里运行。

云（**Cloud**）

这种 Hadoop 环境与其他的相比完全不同，一个云环境由物理上分布在很多不同地点的虚拟服务器组成。最流行的 Hadoop 云服务是亚马逊的 EC2（Elastic Compute Cloud，弹性计算云）。

3.2 网络安全

网络安全是一个详细的话题，此处无法彻底阐述。因此，我们将重点关注一些重要的、常用于保护 Hadoop 集群网络的网络安全主题，其中第一个就是*网络划分*。

3.2.1 网络划分

网络划分是将机器和服务从更大的网络中隔离出来的常见做法。不管我们讨论 Hadoop 集群、Web 服务器、部门桌面工作站还是其他系统，这种做法均适用。可以用两种不同的方法创建一个网段，这两种做法通常同时出现。

第一种方法是*物理*网络划分，使用路由器、交换机、防火墙等设备对网络的一部分进行分割。虽然这些设备运行在 OSI 模型的较高层，但从物理层角度看，网络分隔就是一个网段中的所有设备都物理接入独立于更大网络中其他设备的网络设备。

第二种方法是*逻辑*网络划分。逻辑划分运行在 OSI 模型的较高层，最普遍的是在使用 **IP** 寻址的网络层。在逻辑隔离下，同一网段的设备以某种方式组合到一起。实现这个目标的最普遍的方法是使用子网。例如，一个 Hadoop 集群有 150 个节点，有可能这些节点在逻辑上被分到同一个 /24 子网（即一个掩码为 255.255.255.0 的子网，包含最多 256 个 IP 地址，其中 254 个可用）。使用这种方法在逻辑上组织主机便于管理和安全保护。

网络划分方法中最常用的是，采用物理和逻辑网络划分的混合方法。实现这种混合方法的最常用方式是使用 **VLAN**（**virtual local area network**，*虚拟局域网*）。VLAN 允许多个网络子网共享物理交换机，虽然所有 VLAN 共享一个 2 层网络，但每个 VLAN 都是一个独立的广播域。根据网络交换机或路由器的能力，可能需要将每个物理端口分配到一个 VLAN，或者利用分组标记（packet tagging）在同一个端口运行多个 VLAN。

如前所述，物理隔离和逻辑隔离可以并且经常同时使用。物理和逻辑隔离可能存在于内部和管理环境，该环境中，Hadoop 集群具有一个给定的逻辑子网，并且所有机器都物理连接到同一组专用网络设备（例如 ToR 交换机和汇聚交换机）。

云运行环境中，物理上的网络划分通常要困难得多。云基础设施的设计目标就是要让硬件的位置变得不那么重要，更重要的是能够按需调整服务可用性。一些云环境允许用户选择处于同一位置组的机器，虽然从性能角度看这样当然更好（例如网络延迟更小），但通常无益于安全。同一位置组的机器可能与其他机器共享相同的物理网络。

现在，如果有一个 Hadoop 集群位于自己的网段，应当如何保护该网段呢？这很大程度上要靠网络防火墙和入侵检测与防护系统。

3.2.2 网络防火墙

网络防火墙将 Hadoop 集群从它所在的网络中强制隔离，是一种非常好的方法。防火墙的基本前提是，它被用作不同网段之间网络流量的附加安全层。例如，网络防火墙可能存在于公司内部用户网段和含有互联网站点的网段之间。同样，网络防火墙不太可能出现在一座办公楼同一个部门的两台桌面计算机之间。

表面上看，网络防火墙是不同于路由器、交换机等网络设备的另一种硬件，但也不完全如此。现在的路由器和（多层）交换机通常也具备许多与独立防火墙相同的核心功能。我们将在本节讨论 Hadoop 环境下关于防火墙的几个要点。

网络防火墙的基本功能是，根据网络和传输层属性允许或过滤（丢弃）网络数据包。归根结底就是，根据源和目的 IP、协议类型（如 TCP、UDP、ICMP）、源和目的端口（如果适用）进行过滤决策。网络设备很容易进行这些过滤决策，因为所有这些信息都包含在数据包头中，也就是说，并不必须对载荷数据进行深度包检测。

Hadoop 的基本过滤的价值通常见于 3 种常用类型：进出集群的数据传输、客户端访问（包括终端用户和第三方工具）、管理流量。每个类型都从不同视角考虑，如何用网络防火墙确保 Hadoop 集群和其他一切事物之间的网络路径的安全性。

1.数据传输

第一种类型是数据传输，即数据是如何被接收到集群或者提供到下游系统的。第 10 章将详细讨论从 Hadoop 生态系统的角度出发，如何保护这些流的安全。现在重点关注的是这些数据传输的网络通道和包含的数据类型，从而决定防火墙所需的检查级别。

首先看数据传输的网络通道，常见的选择是通用的文件传输工具，例如 FTP 和 SCP、RDBMS 流（经由 Sqoop）、Flume 等流接收工具产生的数据流。这些常见的通道都有相应的 IP 地址和端口，可以很好地进行网络通信识别，以及创建允许该通信的防火墙规则。了解预期流量是什么样的很重要。哪些机器拥有源数据？数据在导入集群之前是否先要到集群边缘的一台服务器上？哪些机器在接收从集群提取的数据？解决这些问题要求网络防火墙规则大体上能够做到以下几点。

- 允许由特定 FTP 服务器向一个或多个边缘节点的 FTP 流量（本章将会进一步描述）；
- 允许集群中的工作节点通过指定端口连接一个或多个数据库服务器，以发送和接收数据；
- 允许数据通过限定数量的端口，从 Web 服务器集群产生的日志事件中传输到一系列 Flume 代理。

接下来要确定的是数据来源，以及是否需要额外的防火墙检测。例如，若一个上游数据源来自内部业务系统，那么上述防火墙策略就足够了。但若来自一个不可信的源（如互联网上提供的数据），则可能需要深度包检测，以保护集群不受恶意内容的危害。

2.客户端访问

第二种常见的类型是关于客户端访问的。我们将在第 11 章详细讨论这个话题。但从网络防火墙的角度看，重要的是了解并区分客户端与集群交互所使用的方法。一些集群会在“全黑”环境中运行，这意味着不允许任何终端用户活动。这种类型的环境通常会运行持续的 ETL 作业，全自动产生结果数据并报告给下游系统。这种环境下，客户端访问策略只

是简单地阻断一切，保持集群运行和安全只需要数据传输和管理相关的策略。

一种更加普遍的环境是，包含了访问集群的用户、工具和应用程序的混合环境。这种情况下，组织管理是关键。第三方工具运行在什么地方？它们能否被单独放到几个已知的机器？用户从什么地方访问集群？能否要求用户使用边缘节点？自定义应用程序运行在什么地方？网络防火墙处于应用程序和集群之间，还是处于应用程序和用户之间？

3.管理流量

最后一个常见的种类是管理流量，包括管理员用户的登录行为、从集群到外部审计服务器的审计事件流量、从集群到另外一个网络的备份流量等。备份可能是通过 DistCp 进行大量的数据传输，甚至是将 Hive metastore 数据库备份到集群数据中心之外的地方。“*管理流量*”这个词并不是要描述量有多少，而是说该流量与常规客户端产生的流量有所不同。

网络防火墙是集群和外部网络之间的一种很好的安全边界，但如何防止可能主动攻击集群机器的恶意流量呢？对此，我们接下来讨论入侵检测和防御。

3.2.3 入侵检测和防御

上一节介绍了，如何使用网络防火墙作为控制进出 Hadoop 集群所在网络的数据流。虽然这种方法对“正常”的日常流量很有效，但如果发生不那么正常的事件，该怎么办呢？如果一个恶意的攻击者绕过了网络防火墙，并尝试攻击集群中的机器（如缓冲区溢出攻击），会怎样呢？如果是分布式拒绝服务攻击呢？入侵检测和防御系统可以帮助阻止这些类型的攻击。探讨入侵检测和防御设备如何融入集群的系统架构之前，先了解一些关于这些系统的基础知识。

讨论网络安全时，*入侵检测系统*（**IDS**）和*入侵防御系统*（**IPS**）经常被混为一谈。然而，这两个系统处理可疑入侵事件时，扮演的角色是根本不同的。IDS，顾名思义是检测入侵事件，与监控和预警系统属于同一类别。IDS 通常以混杂模式接入交换机，这意味着所有交换机上的网络流量在发送到指定目标端口的同时，也会发送给 IDS。当 IDS 发现一个被怀疑是攻击行为的数据包或者数据流时，就会产生一条警告。警告可能是发给独立的监控系统的事件，也可能只是一封安全管理员订阅的邮件。图 3-1 展示了一个包含 IDS 的网络图，可以看出，IDS 并不处于集群网络与外部网络之间的网络流中。

另一方面，IPS 不仅检测入侵行为，还会在入侵行为发生时主动尝试*预防*或阻止。之所以能做到这一点是因为，IDS 和 IPS 的关键区别在于，IPS 不在网络中以混杂模式监听数据，而处于网络两侧的*中间*。正因如此，IPS 可以阻断流向网络另一端的入侵数据流。IPS 的一个常见功能就是*故障时关闭*（**fail close**），这意味着，IPS 发生故障时（例如遭受大面积的 DDoS 攻击，导致其无法扫描数据包），它会直接阻断*所有*流向 IPS 另一侧的数据包。虽然这看上去可能也算一种成功的 DDoS 攻击，但在一定程度上，这种故障时关闭的机制反而保护了 IPS 后面的所有设备。图 3-2 展示了包含 IPS 的网络图，可以发现，IPS 实际上处于集群网络与外部网络之间的网络流中。

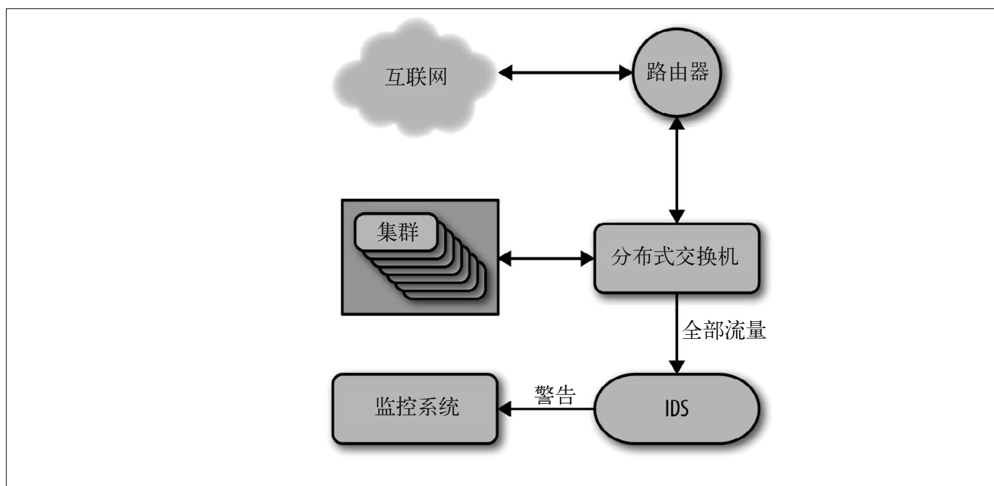


图 3-1：包含 IDS 的网络图

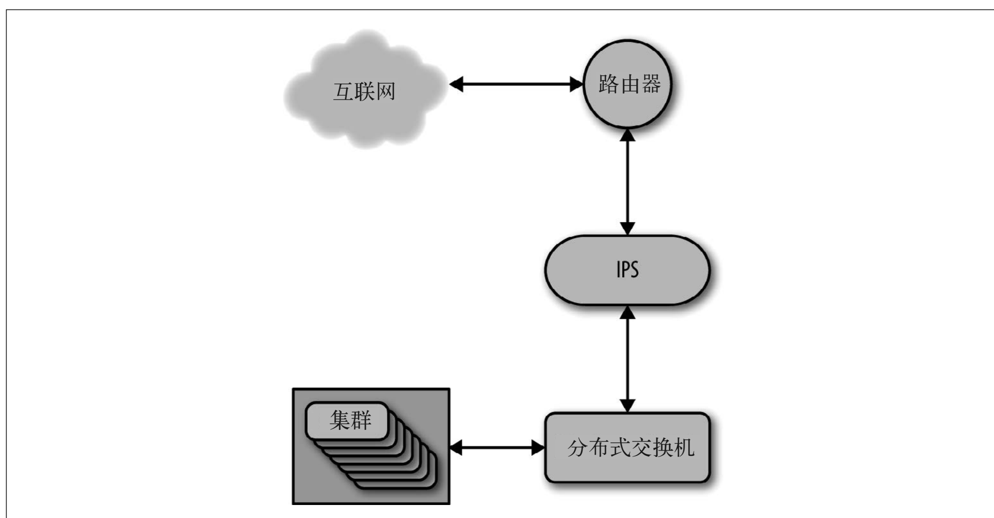


图 3-2：包含 IPS 的网络图

我们已经从宏观上了解了这些设备的作用，这对于 Hadoop 又有什么帮助呢？这其实是另外一个网络安全难题。Hadoop 集群本身存储着大量数据，对集群的入侵尝试进行检测和防御对保护大规模数据而言至关重要。那么，相对于 Hadoop 集群所在网络的其他部分而言，这些设备应当放在什么位置呢？答案是：可能是多个地方。

讨论网络防火墙的时候我们提到过，从安全角度看，来自开放互联网的数据接收通道需要被区别对待。对于 IDS 和 IPS 设备来说也同样如此，入侵攻击大部分来自互联网上的恶意用户。考虑到这一点，将 IPS 放到互联网数据源的接收路径中就十分合理了。互联网可能是恶意攻击者的温床，但企业的内部威胁也是实际存在的，且不容忽视。选择安全架构

时，必须假定恶意攻击者也在同一座楼中工作。将 IDS 放在可信网络内部后，可以作为提醒管理员防范内部威胁的重要工具。

讨论 IDS 和 IPS 的一个额外的好处是，记录大量网络流也是很好的 Hadoop 用例。安全公司常常使用 Hadoop 收集很多不同用户网络中的 IDS 日志，以进行一系列的大规模数据分析和可视化，这也会反过来扩充防火墙和 IDS/IPS 设备使用的规则引擎。

3.3 Hadoop角色和隔离策略

之前我们提到，集群中的节点可以被分成几组，以帮助建立充分的安全策略。本节看看如何做到这一点。每个节点在集群中都扮演着某种角色，这些角色决定了需要哪些安全策略来保护它。首先，回顾一下常见的 Hadoop 生态系统组件，以及每个组件承担的服务角色（我们假定你已经了解这些服务角色是做什么的，如果不是，请快速回顾第 1 章）。

HDFS
NameNode（活跃 / 待命 / 备用）、DataNode、JournalNode、FailoverController、HttpFS、NFSGateway

MapReduce
JobTracker（活跃 / 待命）、TaskTracker、FailoverController

YARN
ResourceManager（活跃 / 待命）、NodeManager、JobHistory Server

Hive
Hive Metastore Server、HiveServer2、WebHCatServer

Impala
Catalog Server、StateStore Server、Impalad

Hue
HueServer、Beeswax、KerberosTicketRenewer

Oozie
OozieServer

ZooKeeper
ZooKeeper Server

HBase
Master、RegionServer、ThriftServer、RESTServer

Accumulo
Master、TabletServer、Tracer、GarbageCollector

Solr
SolrServer

管理和监控服务
: Cloudera Manager、Apache Ambari、Ganglia、Nagios、Puppet、Chef 等

通过这个（非详尽的）列表可以看到，许多生态系统项目都有主 / 从式架构，这很适合从安全架构角度组织服务角色。此外，还有一些服务角色是面向客户端的。总的来说，隔离策略是，确定运行在*主节点*上的主服务、*工作节点*上的工作服务和*管理节点*上的管理服务，还要确定哪些组件需要部署客户端配置文件，从而使得用户能够访问该服务。这些客户端配置文件与面向客户端的服务一起放到*边界节点*上。接下来将更详细地讲解节点的分类。

3.3.1 主节点

主节点可能是最重要的节点组，它包括所有作为 Hadoop 骨干的主要服务。由于这些角色对于它们所代表的组件的重要性，所以也需要更多安全策略保护自己。如下是需要在专用主节点中运行的角色列表：

- HDFS NameNode、备用 NameNode（或者待命 NameNode）、Failover Controller、JournalNode、KMS
- MapReduce JobTracker、FailoverController
- YARN ResourceManager、JobHistory Server
- Hive Metastore Server
- Impala Catalog Server、StateStore Server
- Sentry Server
- ZooKeeper Server
- HBase Master
- Accumulo Master、Tracer、GarbageCollector

有了这个服务列表后，第一个安全问题是：谁需要访问主节点，目的是什么？简单的答案是：管理员，为了执行管理功能。集群的客户——不管是实际的终端用户还是第三方工具——都可以利用开放的标准接口远程访问所有这些服务。例如，一个使用 `hdfs dfs -ls` 命令的用户可以在任何拥有相应 HDFS 服务客户端配置的机器上成功执行该命令，而不需要在运行 HDFS NameNode 的主节点上执行。考虑到这一点，需要将主节点限制为“仅限管理员访问”，原因如下。

资源连接
: 如果普通终端用户能够使用主节点运行任意程序，从而使用系统资源，那么这会夺走原本主节点角色所需的资源，从而导致性能下降。

安全漏洞
: 软件天生就具有漏洞，Hadoop 也不例外。如果允许用户访问担任主节点角色的机器，那就为利用 Hadoop 代码中未修补的漏洞（不管是恶意的还是无意的）打开了一扇大门。限制对主节点的访问降低了利用这些安全漏洞的风险。

拒绝服务
: “用户可以做出不可思议的事情”，这已经是很客气的说法了。如果终端用户使用与主节点角色相同的机器，就不可避免地为其做出某些会搞垮主进程的行为创造条件。回到资

源争用的问题。如果用户运行了一个占满日志目录的失控的进程，结果会怎么样？所有主节点角色都能处理无法记录日志的情况吗？管理员会想找出问题吗？另外一个类似的例子是，一个失控的进程占满了系统的 CPU 或者内存资源，从而导致系统出现内存不足的错误。

3.3.2 工作节点

工作节点负责处理 Hadoop 集群实际做的大部分工作，即存储和处理数据。工作节点上的典型角色如下。

- HDFS DataNode
- MapReduce TaskTracker
- YARN NodeManager
- Impala Daemon
- HBase RegionServer
- Accumulo TabletServer
- SolrServer

由于这些角色负责处理用户的数据和处理请求，所以表面上看，好像所有集群用户都需要访问这些节点，然而并非如此。通常只有管理员才需要远程访问工作节点以执行维护任务，而终端用户可以通过相关的接口和 API 进行数据接收、任务提交和记录检索。大多数情况下，服务会提供一种代理机制，允许管理员将用户活动定向到实际工作节点之外的一系列其他节点。具体细节稍后详细阐述。这些代理代表用户与工作节点进行通信，从而消除了直接访问的必要。

与主节点一样，合理的做法是仅限管理员访问工作节点，理由如下。

资源争用

当普通终端用户在工作节点上的预期进程之外进行活动时，可能造成资源管理失衡。例如，YARN 可被配置成使用特定数量的系统资源，具体数量是 Hadoop 管理员根据操作系统和其他软件的实际需要计算的。但终端用户的活动如何计算？通常很难精确描绘和计算用户活动，因此，很可能使用率高的工作节点相比未被使用的工作节点运行得不是很好，或者无法预测。

工作角色失衡

如果终端用户使用工作节点进行日常活动，可能造成工作节点角色行为上的不良失衡。例如，如果终端用户经常登录某一个特定的、运行着 DataNode 角色的工作节点，那么从该节点接收数据就会产生磁盘使用的失衡。因为 HDFS 写操作将会在选择集群中其他位置之前，先尝试写入本地的第一个区块。这意味着，如果一个用户尝试上传一个 10 GB 的文件到 HDFS 的 home 目录，这 10 GB 都会被写到本地接收数据的 DataNode。

3.3.3 管理节点

管理节点是管理员的命脉。这些节点提供了安装、配置、监控和维护 Hadoop 集群的机制。这些节点上的典型角色如下。

- 配置管理
- 监控
- 警告
- 软件仓库
- 备份数据库

这些管理节点通常包含集群的软件仓库，尤其是在 Hadoop 集群节点没有互联网访问权限的情况下。管理节点最重要的职能是配置管理软件。不管是 Hadoop 专用的（如 Cloudera Manager、Apache Ambari）还是非专用的（如 Puppet、Chef），都是管理员建立和配置集群的地方。配置管理的必然结果是监控和警告，这些角色由 Ganglia、Nagios 以及 Hadoop 专用管理控制台等软件包提供。

此处仍要强调：这些节点不是给普通用户的。Hadoop 集群的管理和维护是一项管理职能，因此也应当被作为管理职能得到保护。尽管如此，也有一些例外。一种常见的例外是，开发者拥有集群监控仪表板的访问权限，从而在作业运行时监控相关指标，确认代码性能水平。

3.3.4　边界节点

边界节点受到所有 Hadoop 集群用户的关心，这些节点包含了最终为用户提供 Hadoop 存储和计算系统利用机制的 Web 接口、代理和客户端配置。边界节点上通常有如下角色。

- HDFS HttpFS 和 NFS 网关
- Hive HiveServer2 和 WebHCatServer
- Impala 网络代理 / 负载均衡器
- Hue Server 和 Kerberos 票据更新器
- Oozie Server
- HBase Thrift Server 和 REST Server
- Flume Agent
- 客户端配置文件

看到边界节点上常见的角色列表时，会明显发现，这种节点类型与其他节点有些不同。与其他节点类型的情况不同的是，各个边界节点通常都大相径庭。例如，使用 Flume 代理的接收管道通常会在用户不能访问的边界节点上，而包含便于命令行访问的客户端配置的边界节点则应当是用户可访问的。“边界节点组内的节点分类需要多么细致”取决于很多因素，包括集群规模和具体用例。以下是对边界节点进行进一步分类的例子。

数据网关

HDFS HttpFS 和 NFS 网关、HBase Thrift Server 和 REST Server、Flume 代理

***SQL* 网关**

Hive HiveServer2 和 WebHCatServer、Impala 负载均衡代理（如 HAProxy）

用户门户

Hue Server 和 Kerberos 票据更新器、Oozie Server、客户端配置文件



虽然 Impala 守护进程不一定必须与 HDFS DataNode 搭配使用，但不建议使用独立的 Impala 守护进程作为代理。更好的选择是，使用负载均衡代理作为负载均衡器，如 HAProxy。当客户由于防火墙或其他限制无法直接连接工作节点上的 Impala 服务时，推荐使用这种架构。

使用上面给出的额外边界节点分类，可以很容易划分出哪些节点上用户理应有对其的远程访问权限，哪些节点只能通过配置的远程端口进行远程访问。虽然用户需要远程访问门户节点，以通过控制台与集群进行交互，但数据和 SQL 网关不能通过这种方式访问是有道理的。这些节点只能通过远程端口访问，以便于访问在用户门户运行的命令行工具，和可能处在网络中其他位置的其他商业智能工具。

以上的分组只是示例，重要的是，不仅要了解安装在集群中的服务，还要知道这些服务是如何使用、被谁使用的。这又绕回之前关于了解用户和运行环境的讨论了。

3.4　操作系统安全

本节将探讨如何在操作系统层保护各个节点。

3.4.1　远程访问控制

典型的服务器环境中，远程访问控制比较简单。比如，一个运行 RDBMS 或 Web 服务的服务器通常是对终端用户锁定的，只允许特权用户以及管理员登录。而 Hadoop 环境没有这么简单。由于 Hadoop 生态系统的内在复杂性，除了基本管理的典型角色和职责之外，还有无数工具和访问方法可以与集群进行交互。虽然 Hadoop 集群可能包括成千上万的节点，但正如本节稍后将讲述的，这些节点是可以被分组的。考虑到这一点，确定哪些机器需要以及为什么需要被访问，并依此限制对机器的远程访问就很重要了。

3.4.2　主机防火墙

远程访问控制是限制哪些用户可以登录集群中特定机器的好方法。这固然是有用和必需的，但只是保护集群中特定机器方法的一小部分。主机防火墙是限制进出节点网络流量类型的一种十分有用的工具。Linux 系统中，主机防火墙通常是利用 iptables 实现的。当然也有其他第三方软件包可以实现该功能（如商业软件），但我们将关注的重点放在 **iptables** 上，因为它在大多数 Linux 发行版中都是默认提供的。

要使用 iptables，首先要对 Hadoop 集群中的网络流量进行理解和分类。表 3-1 展示了 Hadoop 生态系统组件常用的端口。我们将用这张表开始建立 iptables 的主机防火墙策略。

表3-1：Hadoop服务常见端口

组件	服务	端口
Accumulo	Master	9999
	GarbageCollector	50091
	Tracer	12234
	ProxyServer	42424
	TabletServer	9997
	Monitor	4560，50095
Cloudera Impala	Catalog Server	25020，26000
	StateStore	24000，25010
	Daemon	21000，21050，22000，23000，25000，28000
	Llama ApplicationMaster	15000，15001，15002
Flume	Agent	41414
HBase	Master	60000，60010
	REST Server	8085，20550
	Thrift Server	9090，9095
	RegionServer	60020，60030
HDFS	NameNode	8020, 8022, 50070, 50470
	SecondaryNameNode	50090，50495
	DateNode	1004，1006，50010，50020，50075，50475
	JournalNode	8480，8485
	HttpFS	14000，14001
	NFS Gateway	111，2049，4242
	KMS	16000，16001
Hive	Hive Metastore Server	9083
	HiveServer2	10000
	WebHCat Server	50111
Hue	Server	8888
MapReduce	JobTracker	8021，8023，9290，50030
	FailoverController	8018
	TaskTracker	4867，50060
Oozie	Server	11000，11001，11443
Sentry	Server	8038，51000
Solr	Server	8983，8984
YARN	ResourceManager	8030, 8031, 8032, 8033, 8088, 8090
	JobHistory Server	10020，19888，19890
	NodeManager	8040，8041，8042，8044
Zookeeper	Server	2181，3181，4181，9010

有了以上常用端口后，我们应当了解需要实施多么严格的策略。配置 iptables 规则包括端口和 IP 地址，以及通信方向。一个最基本的防火墙策略允许任意主机访问被允许的端口，并且允许所有返回的（已建立连接的）流量。一个 HDFS NameNode 的 iptables 策略可能看起来和示例 3-1 差不多。

示例 3-1　基本的 NameNode iptables 策略

```
iptables -N hdfs
iptables -A hdfs -p tcp -s 0.0.0.0/0 --dport 8020 -j ACCEPT
iptables -A hdfs -p tcp -s 0.0.0.0/0 --dport 8022 -j ACCEPT
iptables -A hdfs -p tcp -s 0.0.0.0/0 --dport 50070 -j ACCEPT
iptables -A hdfs -p tcp -s 0.0.0.0/0 --dport 50470 -j ACCEPT
iptables -A INPUT -j hdfs
```

该策略很宽松，它允许所有主机（0.0.0.0/0）通过常用的 HDFS NameNode 服务端口连接到机器上。然而，这个策略可能太开放了。假定 Hadoop 集群节点都在子网 10.1.1.0/24 中，并且一个用于集群通信的专用边界节点设置在主机 10.1.1.254 上。此外，Web 控制台开启了 SSL。那么这个 NameNode 机器的调整后的 iptables 策略可能会如示例 3-2 所示。

示例 3-2　安全的 NameNode iptables 策略

```
iptables -N hdfs
iptables -A hdfs -p tcp -s 10.1.1.254/32 --dport 8020 -j ACCEPT
iptables -A hdfs -p tcp -s 10.1.1.254/32 --dport 8022 -j DROP
iptables -A hdfs -p tcp -s 10.1.1.0/24 --dport 8022 -j ACCEPT
iptables -A hdfs -p tcp -s 0.0.0.0/0 --dport 50470 -j ACCEPT
iptables -A INPUT -j hdfs
```

调整后的策略要严格得多。它允许任意用户通过 SSL（50470 端口）连接到 NameNode Web 控制台，仅允许集群机器通过专用的 DataNode RPC 端口（8022）连接到 NameNode，并且仅允许来自边界节点的流向 NameNode RPC 端口（8020）的用户流量。



为了使策略生效，可能需要将 iptables 跳转目标插入 `INPUT` 部分的特定行号。此处为了简便，仅展示了追加（Append）操作。

3.4.3　SELinux

另外一个经常被讨论的关于操作系统安全的内容是*安全增强型* **Linux**（**SELinux**），最早由美国的情报机构——美国国家安全局（NSA）开发。SELinux 的前提是提供 Linux 内核增强，从而允许*强制访问控制*（**MAC**）的策略和实施。SELinux 大致可以配置为以下几种方式。

禁用（**Disabled**）

　这种模式下，SELinux 不生效，也不给操作系统提供任何额外的安全级别。这是 Hadoop 中极为普遍的配置方式。

许可（**Permissive**）

这种模式下，SELinux 是启用状态，但不对系统进行保护，而在策略被违反时打印警告。对于从了解系统中工作负载类型入手，以建立自定义策略来说，这种模式很有用。

强制（**Enforcing**）

这种模式下，SELinux 是启用状态，并根据特定的 SElinux 策略对系统进行保护。

除了许可和强制两种启用模式之外，SELinux 还有两种不同的强制实施类型：针对性强制实施（targeted enforcement）和多级安全（**multilevel security**，**MLS**）。针对性强制实施仅针对特定进程，这意味着，这些进程具有相关策略进行保护，没有策略的进程则不会被 SELinux 保护。这种保护模式显然不那么严格。另一方面，MLS 则更为深入。总体而言，MLS 的前提是所有用户和进程都具有一个安全级别，同时，文件和其他对象都具有一个安全级别需求。MLS 仿照美国政府的分级标准，如绝密（Top Secret）、机密（Secret）、秘密（Confidential）和非密（Unclassified）。美国政府的分级系统中，这些级别形成一种等级架构。这种架构中，具有特定访问级别的用户具有访问较低级别信息的权限。例如，如果一个用户具有机密安全级，那么该用户被允许访问操作系统中机密、秘密和非密的对象——因为秘密和非密的安全级别都比机密低，但该用户无法访问标记为“绝密”安全级的对象。

这些听起来很好，但与 Hadoop 有什么关系呢？ SELinux 能否被用作运行各种 Hadoop 生态系统组件的操作系统的额外保护层呢？简短的回答是：不太能。并不是说不可能，而是承认在这一点上缺乏 SELinux 安全集成以及相关（可被安全管理员部署到集群中的）策略创建方面的发展。Hadoop 生态系统的本质才是问题所在。目前有数以百计的组件、工具和其他部件以这样那样的方式集成到平台，并 / 或对平台进行增强。加进来的工具越多，想出一系列 SELinux 策略对其进行全部管理的难度就越大。

如果非要这么用，可能的方法是，将系统设置为许可模式，并在集群中运行相当于“普通”级别的、使用了尽可能多的给定环境中的典型工具的工作任务。这样运行一段合适的时间后，即可使用 SELinux 生成的警告开始建立策略。此处的问题是，这很快会变成一个枯燥乏味的过程，而且每当引入新的组件或功能时，都要对该过程进行修改。

3.5　小结

本章从定义 Hadoop 所在的操作系统环境开始，粗略地分析了 Hadoop 环境。然后从网络安全角度讨论了对该环境的保护，包括利用常见的安全实践（如网络划分）、介绍防火墙和 IDS/IPS 等网络安全设备。随后了解了如何根据运行的服务类型将 Hadoop 集群分成不同的节点组。最后给出了根据节点分组保护单个节点操作系统的建议。

第 4 章将介绍 Hadoop 安全架构中的一个基础组件：Kerberos。Kerbero 是企业系统中的一个关键成员，Hadoop 也不例外。下一章将会结束关于安全架构的讨论，为认证、授权和审计奠定基础。

第4章
Kerberos

第一次提 Kerberos 的时候，甚至那些有经验的系统管理员和开发者都会被吓到。依赖 Kerberos 的应用程序和系统经常会有很多需要解决相关问题的求助电话和故障单。本章将介绍基本的 Kerberos 概念，这对理解强认证是如何工作的很有必要；也会阐述 Kerberos 在第 5 章中的 Hadoop 认证中如何发挥重要作用。

那么 Kerberos 究竟是什么？在神话里，Kerberos 是 **Cerberus** 的希腊语，是一只守护地狱入口的三头巨犬，它确保没有人能在进入地狱后离开。从技术角度（这个角度更轻松一些）来说，Kerberos 是麻省理工学院开发的一个认证机制。Kerberos 发展成为大大小小的计算机系统强认证的实际标准，具有很多不同的实现，包括从 MIT 的 Kerberos 分发到微软的活动目录认证组件。

4.1 为什么是Kerberos

为什么 Hadoop 需要 Kerberos？看看 Hadoop 认证的默认模型，原因就很明显了。提交给 Hadoop 一个用户名时，它会很高兴地相信你所说的一切，并且确保整个集群的所有其他机器也相信。

打个比方，如果在聚会中，一个人接近你并自我介绍为 Bill，你自然会相信他确实是 Bill。你怎么知道他确实是 Bill？因为他是这么说的，而且你毫无疑问地相信了他。缺少 Kerberos 的 Hadoop 差不多也是这样，不同的是它还做了进一步类推，不仅相信他就是他自己所说的 Bill，还保证其他人也都相信。这是个问题。

Hadoop 的设计初衷是存储和处理 PB 级的数据。老话说得好，“能力越大，责任越大”。企业中的 Hadoop 不能再使用过于简单方法来识别（和信任）用户了。之前的比喻中，Bill 向你介绍了他自己。这时候，如果你要求看他的有效证件，并在收到证件之后（这很自

然，因为每个人参加聚会时都会带证件……）利用数据库验证其有效性，会怎样呢？这就和 Hadoop 通过添加 Kerberos 认证引入的身份验证是一样的。

4.2 Kerberos概览

万事俱备，现在深入讨论和理解 Kerberos 是如何工作的。正如你可能想到的那样，Kerberos 的实现是一种客户端 / 服务端架构。对 Kerberos 组成部分进行详细分解之前，先介绍一些 Kerberos 术语。

首先，Kerberos 中的身份标识称为*主体*（**principal**），每个参与 Kerberos 认证协议的用户和服务都需要一个主体来唯一地标识自己。主体分为两种：*用户主体*和*服务主体*。*用户主体名称*（**user principal name**，**UPN**）代表常规用户，类似于操作系统中的用户名或者账号。*服务主体名称*（**service principal name**，**SPN**）代表用户需要访问的服务，例如特定服务器上的数据库。稍后会通过一个例子更清楚地说明 UPN 和 SPN 的关系。

接下来一个重要的 Kerberos 术语是*域*（**realm**）。一个 Kerberos 域就是一个身份验证管理域，所有主体都被分配到特定的 Kerberos 域。域确立了边界，这使得管理更为容易。

确定了什么是主体和域之后，下一步自然是弄清楚存储和控制这些信息的是什么。答案是*密钥分发中心*（**key distribution center**，**KDC**）。KDC 由三部分组成：Kerberos 数据库、*认证服务*（**authentication service**，**AS**）和*票据授予服务*（**ticket-granting service**，**TGS**）。Kerberos 数据库存储的内容包含主体及其所属域的所有相关信息，数据库中的 Kerberos 域使用如下命名约定进行标识。

`alice@EXAMPLE.COM`
: 唯一标识了 Kerberos 域 `EXAMPLE.COM` 中用户（也叫作*短名称*）`alice` 的 UPN。依照惯例，域名总为大写。

`bob/admin@EXAMPLE.COM`
: 一种常规 UPN 的变形，标识了域 `EXAMPLE.COM` 中一个管理员 `bob`。UPN 中的斜杠（`/`）用于分隔短名称和管理员标识。通常会约定使用 `admin` 表示管理员，但稍后会看到，这也是可以配置的。

`hdfs/node1.example.com@EXAMPLE.COM`
: 该主体表示一个 `hdfs` 服务的 `SPN`，该服务在 Kerberos 域 `EXAMPLE.COM` 中的主机 `node1.example.com` 上。SPN 中的斜杠（`/`）用于分隔短名称 `hdfs` 和主机名 `node.example.com`。



整个主体名都是大小写敏感的。例如 `hdfs/Node1.Hadoop.com@EXAMPLE.COM` 和第三个例子中的主体是不一样的。一般情况下，最好在主体的域部分使用大写，其他部分使用小写。值得注意的是，SPN 中涉及的主机名当然也是小写的，在主机命名和 DNS（域名解析）时最好也这样。

KDC 的第二部分——AS——负责在客户端向 AS 发起请求时，向客户端发放票据授予票据。TGT 用于请求访问其他服务。

KDC 的第三部分——TGS——负责验证 TGT，并授予*服务票据*（**service tickets**）。服务票

据允许认证过的主体使用应用服务器提供的服务，该服务通过 SPN 进行标识。下一节将介绍获得 TGT、提交给 TGS 以及获取服务票据的流程，现在只需了解 KDC 有两部分：AS 和 TGS，负责处理认证和访问服务的请求。



Kerberos 数据库中有一种特种格式的主体 `krbtgt/<REALM>@<REALM>`，例如 `krbtgt/EXAMPLE.COM@EXAMPLE.COM`。该主体是 AS 和 TGS 内部使用的，其关键之处在于，它被用来加密发放给客户端的 TGT 内容，这保证了 AS 发放的 TGT 只能被 TGS 验证。

表 4-1 简要提供了本章涉及的 Kerberos 术语及其缩写。

表4-1：Kerberos术语缩写

术语	名称	描述
UPN	用户主体名	定义给定域中一个用户的主体，格式为 <短名称><@域> 或 <短名称>/admin@<域>
SPN	服务主体名	定义给定域中特定主机上一个服务的主体，格式为 <短名称>/<主机名>@<域>
TGT	票据授予票据	AS 认证成功后授予用户的一种特殊票据
KDC	密钥分发中心	一种 Kerberos 服务器，包含 3 部分：Kerberos 数据库、AS 和 TGS
AS	认证服务	分发 TGT 的 KDC 服务
TGS	票据授予服务	验证 TGT 和授予服务票据的 KDC 服务

之前展示的是一些基本的 Kerberos 组件，这些组件是粗略理解认证机制所需的。Kerberos 本身是一个很深入和复杂的话题，值得用一整本书介绍。好在已经有人这么做了，如果你想更深入地了解 Kerberos，可以参考 Jason Garman 的著作 *Kerberos: The Definitive Guide*。

4.3 Kerberos工作流：一个简单示例

现在通过一个工作流示例，展示 Kerberos 大概是怎么工作的。首先定义所有出现的组件。

`EXAMPLE.COM`
　Kerberos 域。

Alice
　一个系统用户，其 UPN 为 `alice@EXAMPLE.COM`。

`myservice`
　`server1.example.com` 上运行的一个服务，其 SPN 为 `myservice/server1.example.com@EXAMPLE.COM`。

`kdc.example.com`
　Kerberos 域 `EXAMPLE.COM` 的 KDC。

Alice 要想使用 `myservice`，需要向 `myservice` 提供一个有效的服务票据，具体方法如下所述（为了简便，此处省略部分细节）。

(1) Alice 需要获取一个 TGT。为此，她向 kdc.example.com 上的 AS 发起一个请求，表明自己是主体 alice@EXAMPLE.COM。
(2) AS 做出响应，为主体 alice@EXAMPLE.COM 提供一个使用密钥（密码）加密的 TGT。
(3) 接收到加密信息后，Alice 被提示输入主体 alice@EXAMPLE.COM 的正确密码，从而解密信息。
(4) 成功解密包含 TGT 的信息后，Alice 向 kdc.example.com 上的 TGS 请求服务 myservice/server1.example.com@EXAMPLE.COM 的一个服务票据，并在请求中提供出 TGT 信息。
(5) TGS 对 TGT 进行验证，并提供给 Alice 一个使用主体 myservice/server1.example.com@EXAMPLE.COM 的密钥加密的服务票据。
(6) Alice 现在将服务票据提供给 myservice，myservice 随后使用 myservice/server1.example.com@EXAMPLE.COM 的密钥对其解密，并验证票据有效性。
(7) 由于 Alice 已经正确验证其身份，因此服务 myservice 允许其使用。

以上大致描述了 Kerberos 是如何工作的。显然，这是一个大大简化了的例子，很多底层的细节没有得到展示。图 4-1 展示了这个例子的序列图。

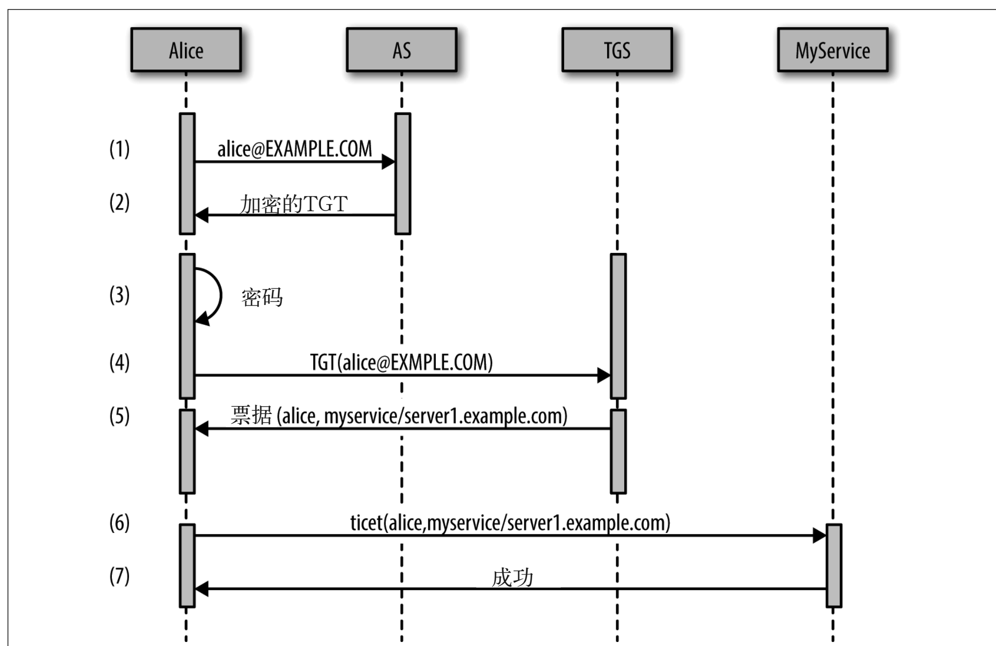


图 4-1：Kerberos 工作流示例

4.4 Kerberos信任

目前为止，我们介绍的 Kerberos 都默认所有用户和服务处于同一 Kerberos 域内。虽然这很适合入门，但考虑到庞大的公司规模，这通常是不符合实际的。随着时间的发展，大公司最终通过合并、收购，或仅仅想分隔公司的不同部分，将形成多个 Kerberos 域。然而，

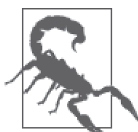
KDC 默认情况下只知道自身的域，以及自身数据库包含的主体。那么，当一个域中的用户想要使用另外一个域控制下的服务，该怎么办呢？为了实现这个目标，两个域之间需要 Kerberos 信任（**trust**）。

例如，假定 Example 是一个很大的公司，并且决定创建多个域表示不同业务线，包括 `HR.EXAMPLE.COM` 和 `MARKETING.EXAMPLE.COM`。因为这两个域的用户有可能都需要访问两个域的服务，因此 `HR.EXAMPLE.COM` 的 KDC 需要信任来自 `MARKETING.EXAMPLE.COM` 域的信息，反之亦然。

表面上看起来很简单，但其实信任有两种类型：单向信任（**one-way trust**）和双向信任（**two-way trust**，或者 **bidirectional trust**、**full trust**）。我们刚才看的例子是一种双向信任。

如果还有一个 `DEV.EXAMPLE.COM` 域，其中的开发者主体需要访问 `DEV.EXAMPLE.COM` 和 `MARKETING.EXAMPLE.COM` 域，而营销用户不能访问 `DEV.EXAMPLE.COM` 域呢？这种场景就需要单向信任。单向信任在 Hadoop 部署中很常见，KDC 被安装配置成包含集群节点 SPN 的所有信息，但所有终端用户的 UPN 都在另外一个域，例如 Active Directory（活动目录）中。通常，Active Directory 管理员或公司政策会出于种种原因禁止双向信任。

那么 Kerberos 信任是如何实际建立的呢？本章之前提到过，有一种特殊的主体被 AS 和 TGS 内部使用，其格式是 `krbtgt/<域>@<域>`。这个主体在建立信任时更为重要，此时，该主体的格式变为 `krbtgt/<信任域>@<被信任域>`。该主体的关键之处在于，它在两个域中都存在。例如，如果 `HR.EXAMPLE.COM` 域需要信任 `MARKETING.EXAMPLE.COM` 域，那么主体 `krbtgt/HR.EXAMPLE.COM@MARKETING.EXAMPLE.COM` 需要在两个域中都存在。



`krbtgt/<信任域>@<被信任域>` 主体的密码和加密类型必须与两个域中的都一样，以便建立信任。

上一个例子展示了单向信任中需要什么。为了建立双向信任，`krbtgt/MARKETING.EXAMPLE.COM@HR.EXAMPLE.COM` 主体也需要在两个域中都存在。总之，为了使 `HR.EXAMPLE.COM` 域和 `MARKETING.EXAMPLE.COM` 域具有双向信任，两个域都需要有 `krbtgt/MARKETING.EXAMPLE.COM@HR.EXAMPLE.COM` 和 `krbtgt/HR.EXAMPLE.COM@MARKETING.EXAMPLE.COM` 主体。

4.5 MIT Kerberos

正如本章开始提到的，Kerberos 由 MIT 首先创造。这些年来，它已历经数次修改，当前最新的版本是 `MIT Kerberos V5`，常称为 `krb5`。本节将介绍 MIT Kerberos 发行版的几个组件，用几个实际案例实践之前介绍的概念性例子。



要想获得关于 MIT Kerberos 发行版的最新权威资源，官方网站有很好的文档可以参考（http://web.mit.edu/~kerberos/）。

早先的例子里，我们忽略了一件事——Alice 发起了认证请求。实际上，Alice 是使用 **kinit** 工具做到这一点的。

示例 4-1　kinit（使用默认用户）

```
[alice@server1 ~]$ kinit
Enter password for alice@EXAMPLE.COM:
[alice@server1 ~]$
```

该例子将当前的 Linux 用户名 `alice` 和默认域合到一起，组成建议的主体 `alice@EXAMPLE.COM`。稍后深入讨论配置文件时，我们会讲解默认域。`kinit` 工具也允许用户显式地定义主体，以进行认证，如示例 4-2 所示。

示例 4-2　kinit（使用特定用户）

```
[alice@server1 ~]$ kinit alice/admin@EXAMPLE.COM
Enter password for alice/admin@EXAMPLE.COM:
[alice@server1 ~]$
```

如上所示，认证为管理员用户时，显式地提供主体名通常是必要的。另一个认证方法是使用 **keytab** 文件。keytab 文件存储了可被用于代替密码的加密密钥。创建 keytab 文件对于非交互式主体很有用，例如与长期运行着的进程（如 Hadoop 服务）相关的 SPN。keytab 文件并不需要一一映射到每个主体，多个不同主体的密钥可存储在一个 keytab 文件里。用户可以在使用 `kinit` 的时候指定 keytab 文件的路径，以及要认证为的主体名称（因为 keytab 文件中可能存在多个主体），如示例 4-3 所示。

示例 4-3　kinit（使用 keytab 文件）

```
[alice@server1 ~]$ kinit -kt alice.keytab alice/admin@EXAMPLE.COM
[alice@server1 ~]$
```



keytab 文件允许用户在没有密码相关知识的前提下进行认证。正因如此，keytab 需要用适当的控制手段加以保护，防止未授权用户使用其进行认证。keytab 是为管理员主体创建的时候，这尤为重要。

MIT Kerberos 发行版的另一个有用的功能称为 `klist`。该功能允许用户查看他们的*证书缓存*（**credentials cache**）中的 Kerberos 证书（如果有）。证书缓存存放在本地文件系统中，与成功通过 AS 验证的 TGT 的存储位置相同。默认情况下，该位置通常是 `/tmp/krb5cc_<uid>` 文件，其中 `<uid>` 是本地系统用户 ID 号。成功执行 `kinit` 后，alice 可以使用 `klist` 查看她的证书缓存，如示例 4-4 所示。

示例 4-4　使用 klist 查看证书缓存

```
[alice@server1 ~]$ kinit
Enter password for alice@EXAMPLE.COM:
[alice@server1 ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_5000
Default principal: alice@EXAMPLE.COM

Valid starting     Expires            Service principal
```

```
02/13/14 12:00:27  02/14/14 12:00:27  krbtgt/EXAMPLE.COM@EXAMPLE.COM
        renew until 02/20/14 12:00:27
[alice@server1 ~]$
```

如果用户在没有进行认证的情况下尝试查看凭证缓存，则找不到任何凭证。

示例 4-5　找不到凭证

```
[alice@server1 ~]$ klist
No credentials cache found (ticket cache FILE:/tmp/krb5cc_5000
[alice@server1 ~]$
```

MIT Kerberos 工具箱中另一个有用的工具是 kdestroy。顾名思义，它允许用户销毁凭证缓存中的凭证。这在切换用户、尝试或调试新配置的时候很有用（示例 4-6）。

示例 4-6　用 kdestroy 销毁凭证缓存

```
[alice@server1 ~]$ kinit
Enter password for alice@EXAMPLE.COM:
[alice@server1 ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_5000
Default principal: alice@EXAMPLE.COM

Valid starting     Expires            Service principal
02/13/14 12:00:27  02/14/14 12:00:27  krbtgt/EXAMPLE.COM@EXAMPLE.COM
        renew until 02/20/14 12:00:27
[alice@server1 ~]$ kdestroy
[alice@server1 ~]$ klist
No credentials cache found (ticket cache FILE:/tmp/krb5cc_5000
[alice@server1 ~]$
```

到此为止，MIT Kerberos 的示例展示了其是“能用”的。隐藏在这些示例之外的是，为了使其真正工作起来，不管在客户端还是服务端都还需要很多必要的配置。接下来将介绍基本配置，从而把之前介绍的一些概念联系起来。

4.5.1　服务端配置

Kerberos 服务端配置主要在 kdc.conf 文件中，如示例 4-7 所示。该文件在 Red Hat/CentOS 系统中的位置是 /var/kerberos/krb5kdc/。

示例 4-7　kdc.conf

```
[kdcdefaults]
 kdc_ports = 88
 kdc_tcp_ports = 88

[realms]
 EXAMPLE.COM = {
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  supported_enctypes = aes256-cts:normal aes128-cts:normal arcfour-hmac-md5:normal
  max_renewable_life = 7d
}
```

配置文件的第一节 kdcdefaults 包含的配置适用于下面列出的所有域，除非某个域的配置包含同样配置项的值。配置项 kdc_ports 和 kdc_tcp_ports 分别指定了 KDC 监听的 UDP 和 TCP 端口。下一节 realms 则包含了使用该 KDC 作为服务器的所有域。一个 KDC 可以支持多个域。本例中，域的配置项包括如下内容。

acl_file
: 该项指定了管理服务用于进行访问控制的文件位置（稍后会详细介绍）。

dict_file
: 该项指定了包含不允许用作密码的单词的文件，这些密码是容易被破解 / 猜解的。

supported_enctypes
: 该项指定了 KDC 支持的所有加密类型。与 KDC 交互时，客户端必须支持该项列出的加密类型中的至少一种。谨慎使用 DES 等弱加密类型，因为这种加密很容易被攻破。

max_renewable_life
: 该项指定了票据可更新的最长时间。客户端可请求一个不超过该长度的更新有效期。典型的值是 7 天，用 7d 表示。



默认情况下，MIT Kerberos 中的加密选项通常被设成多个加密类型，其中包括 DES 等弱加密类型。可能的话，尽量删掉弱加密类型，以尽可能保证安全。弱加密类型容易被攻破，这已经被证明了。使用 AES-256 时，需要在集群的所有节点安装 Java Cryptographic Extensions，以允许不限强度加密类型。值得注意的是，一些国家禁止使用这些加密类型，请遵守你所在国家管理加密强度的法律。关于加密的更详细讨论参见第 9 章。

acl_file 所在位置（通常是 kadm5.acl 文件）用于控制哪些用户具有管理 Kerberos 数据库的权限。Kerberos 数据库管理员由两条不同但相关的组件控制：kadmin.local 和 kadmin。前者是允许 KDC 服务器的 root 用户修改 Kerberos 数据库的实用工具，顾名思义，它只能够由 Kerberos 数据库所在机器的 **root** 用户执行。想要远程管理 Kerberos 数据库的管理员必须使用 kadmin 服务器。

Kadmin 服务器是一个允许远程连接并管理 Kerberos 数据库的守护进程，这也是 kadm5.acl 文件（示例 4-8）发挥作用的地方。Kadmin 工具使用 Kerberos 认证，而 kadm5.acl 文件则指定了哪些 UPN 被允许执行特权功能。

示例 4-8　kadm5.acl

```
*/admin@EXAMPLE.COM      *
cloudera-scm@EXAMPLE.COM *     hdfs/*@EXAMPLE.COM
cloudera-scm@EXAMPLE.COM *     mapred/*@EXAMPLE.COM
```

这允许来自 EXAMPLE.COM 的带有 /admin 标识的任意主体执行任意管理操作。虽然将 admin 标识改成其他任意名称也可以接受，但从简单性和可维护性起见，还是建议保持这种约定。管理员用户应当只使用自己的管理凭证进行具体的特许操作，就像在 Linux 中，管理员不能使用 **root** 用户的凭证进行日常非管理操作一样。

上例也说明，ACL 可以如何被定义以限制特定主体的权限。它展示了用户 `clouderascm` 可以执行任意操作，但仅限于在以 `hdfs` 和 `mapred` 开头的 SPN 上。这种语法类型有助于向第三方工具授予创建和管理 Hadoop 主体的权限，但不授予执行所有管理功能的权限。

如前所述，`kadmin` 工具允许 Kerberos 数据库的管理，该工具带给用户一个类似于控制台的接口，以输入各种命令和对 Kerberos 数据库进行操作（示例 4-9 ～示例 4-12）

示例 4-9　向 Kerberos 数据库添加一个新主体

```
kadmin: addprinc alice@EXAMPLE.COM
WARNING: no policy specified for alice@EXAMPLE.COM; defaulting to no policy
Enter password for principal "alice@EXAMPLE.COM":
Re-enter password for principal "alice@EXAMPLE.COM":
Principal "alice@EXAMPLE.COM" created.
kadmin:
```

示例 4-10　显示 Kerberos 数据库中一个主体的详细信息

```
kadmin: getprinc alice@EXAMPLE.COM
Principal: alice@EXAMPLE.COM
Expiration date: [never]
Last password change: Tue Feb 18 20:48:15 EST 2014
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Tue Feb 18 20:48:15 EST 2014 (root/admin@EXAMPLE.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 2
Key: vno 1, aes256-cts-hmac-sha1-96, no salt
Key: vno 1, aes128-cts-hmac-sha1-96, no salt
MKey: vno1
Attributes:
Policy: [none]
kadmin:
```

示例 4-11　删除 Kerberos 数据库中的一个主体

```
kadmin: delprinc alice@EXAMPLE.COM
Are you sure you want to delete the principal "alice@EXAMPLE.COM"? (yes/no): yes
Principal "alice@EXAMPLE.COM" deleted.
Make sure that you have removed this principal from all ACLs before reusing.
kadmin:
```

示例 4-12　列出 Kerberos 数据库中所有主体

```
kadmin: listprincs
HTTP/server1.example.com@EXAMPLE.COM
K/M@EXAMPLE.COM
bob@EXAMPLE.COM
flume/server1.example.com@EXAMPLE.COM
hdfs/server1.example.com@EXAMPLE.COM
hdfs@EXAMPLE.COM
hive/server1.example.com@EXAMPLE.COM
```

```
hue/server1.example.com@EXAMPLE.COM
impala/server1.example.com@EXAMPLE.COM
kadmin/admin@EXAMPLE.COM
kadmin/server1.example.com@EXAMPLE.COM
kadmin/changepw@EXAMPLE.COM
krbtgt/EXAMPLE.COM@EXAMPLE.COM
mapred/server1.example.com@EXAMPLE.COM
oozie/server1.example.com@EXAMPLE.COM
yarn/server1.example.com@EXAMPLE.COM
zookeeper/server1.example.com@EXAMPLE.COM
kadmin:
```

4.5.2　客户端配置

默认的 Kerberos 客户端配置文件通常被命名为 `krb5.conf`，在 UNIX/Linux 系统的 `/etc/` 目录下。客户端应用（包括 `kinit` 工具）需要使用 Kerberos 时，随时读取该配置文件。示例 4-13 展示的 `krb5.conf` 配置文件是根据 Red Hat/CentOS 6.4 中自带配置文件执行的最小配置。

示例 4-13　krb5.conf

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = DEV.EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 default_tkt_enctypes = aes256-cts aes128-cts
 default_tgs_enctypes = aes256-cts aes128-cts
 udp_preference_limit = 1

[realms]
 EXAMPLE.COM = {
 kdc = kdc.example.com
  admin_server = kdc.example.com
 }

 DEV.EXAMPLE.COM = {
   kdc = kdc.dev.example.com
   admin_server = kdc.dev.example.com
 }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .dev.example.com = DEV.EXAMPLE.COM
 dev.example.com = DEV.EXAMPLE.COM
```

该例中有几个不同的节。第一个是 `logging`，其含义不言而喻。它定义了会产生日志的各种 Kerberos 组件的日志文件存放位置。第二节是 `libdefaults`，包括通用的默认配置信息。本节更深入地看一下个性化配置。

`default_realm`
: 该字段定义了没有提供任何域的情况下应该使用什么 Kerberos 域，这与早先 `kinit` 例子中没有提供域的情况刚好是相符的。

`dns_lookup_realm`
: 可以用 DNS 指定使用哪个 Kerberos 域。

`dns_lookup_kdc`
: 可以用 DNS 寻找 KDC 的位置。

`ticket_lifetime`
: 该项指定了票据持续的时间，可以为 KDC 指定的最大值之内的任意时间长度。常用值是 24 小时，标为 `24h`。

`renew_lifetime`
: 该项指定了票据的可*更新*时间。票据可以在不进行客户端认证的情况下，由 KDC 进行更新。这必须在票据过期之前进行。

`forwardable`
: 该项指定了票据是否*可以转发*。这意味着，如果一个用户已经拥有一个 TGT，但登录到其他远程系统，那么 KDC 可以在无需重新认证的情况下向其重新分发一个 TGT。

`default_tkt_enctypes`
: 该项指定了向 AS 发送请求时加密会话密钥的方式。从左到右优先级依次降低。

`default_tgs_enctypes`
: 该项指定了向 TGS 发送请求时加密会话密钥的方式。从左到右优先级依次降低。

`udp_preference_limit`
: 该项指定了 UDP 包的最大值，超过该值后将切换为 TCP。要想强制使用 TCP，应将该值设为 1。

下一节是 `realms`，列出了客户端知道的所有 Kerberos 域。`kdc` 和 `admin_server` 配置项分别告诉客户端哪个服务器运行着 KDC 和 kadmin 进程。这些配置可以在主机名后指定端口，如果没有指定，就默认使用 88（KDC）和 749（admin_server）端口。该例展示了两个域。这是一种常用配置，此时，两个域之间存在单向信任，并且都需要被客户端知晓。该例中，可能 `EXAMPLE.COM` 域包含所有终端用户主体，`DEV.EXAMPLE.COM` 包含一个开发集群中的所有 Hadoop 服务主体。使用这种方式配置 Kerberos 时，能够使该开发集群的用户使用他们在 `EXAMPLE.COM` 域中已有的凭证，以访问 `DEV.EXAMPLE.COM` 域。

最后一节是 `domain_realm`，定义了 Kerberos 域与 DNS 名称的对应关系。第一项表示 `example.com` 域名下的所有主机都映射到 `EXAMPLE.COM` 域，第二项表示 `example.com` 自身映射到 `EXAMPLE.COM` 域。`dev.example.com` 和 `DEV.EXAMPLE.COM` 的情况与之类似。如果本节中没有找到匹配项，客户端就会尝试使用 DNS 名称的域名部分（转换为大写）作为域名。

4.6 小结

本章要点是，Kerberos 认证是一种多级的客户端 / 服务端过程，用以提供用户和服务的强认证。我们介绍了 MIT Kerberos 发行版，它是 Kerberos 的一种主流实现。即便本章描述了如何配置 MIT Kerberos 的一些细节，我们仍然强烈推荐你参考 MIT Kerberos 官方文档（http://web.mit.edu/~kerberos/），因为它是对最新版本的最新参考文档，也为安全管理员配置 Kerberos 环境提供了关于所有配置项的更详细的指南。

第 5 章将把目前为止涉及的 Kerberos 概念放入 Hadoop 以及 Hadoop 生态系统环境，进行更深的探索。

第二部分

验证、授权和审计



第 5 章

身份和验证

对于任何系统，保护数据的第一个必要步骤就是为每位用户提供一个唯一的身份，并且对用户声明的身份进行验证。验证和身份极其重要，因为如果身份验证体系不能够确信用户确实符合他们所声明的身份，则无法对数据进行访问控制。

本章将详细研究 Hadoop 服务如何管理身份验证和身份。首先研究身份的概念，以及 Hadoop 如何合并来自 Kerberos KDC、LDAP 和 Active Directory 域的信息，并提供一个关于分布式身份的综合视图。然后讲解 Hadoop 内部如何表征用户，以及从外部、全局的身份到内部用户角色的映射选项。接下来，回顾 Kerberos，并进一步详细研究 Hadoop 如何通过 Kerberos 实现强认证。之后，再继续研究一些核心组件如何使用基于用户名 / 密码组合的身份验证体系，以及分布式身份验证令牌在整个架构中是怎样的角色。最后，讨论用户模拟，并深入了解 Hadoop 身份验证机制的配置。

5.1 身份

Hadoop 生态系统环境中，身份是一个相对复杂的主题。这是由于，Hadoop 在尽可能地实现与权威身份源的松耦合。第 4 章介绍了 Kerberos 身份验证协议，接下来将对其进行重点介绍，因为 Kerberos 协议是 Hadoop 中默认使用的安全认证协议。虽然 Kerberos 为可靠的身份验证提供支持，但它对高级身份特征（如用户组或角色）几乎没有提供任何支持。特别是 Kerberos 只将身份表现为简单的、分为两部分的字符串（对于服务来说是分为三部分的字符串），这两部分分别为短名称和域。当赋予每位用户一个唯一的身份标识时，这种身份表现法虽然有用，但对于实现可靠的身份验证协议来说仍然不够。

除了用户身份之外，大多数计算系统还提供了组。组通常被定义为一批用户的集合。由于 Hadoop 的目标之一是集成已有的企业系统，所以它务实地使用可插拔系统提供传统的群组概念。

5.1.1 将Kerberos主体映射为用户名

深入了解 Hadoop 如何将用户映射到群组之前，需要讨论 Hadoop 如何将 Kerberos 主体名称转换为用户名。回忆第 4 章中 Kerberos 使用一个两部分字符串（如 `alice@EXAMPLE.COM`）或三部分字符串（如 `hdfs/namenode.example.com@EXAMPLE.COM`），这类字符串包含了短名称、域，以及一个可选的实例名或主机名。为了简化用户名的使用，Hadoop 将 Kerberos 主体名映射至本地用户名。Hadoop 能够使用 `krb5.conf` 文件中的 `auth_to_local` 设置，也能够使用 `core-site.xml` 文件中的 `hadoop.security.auth_to_local` 参数对 Hadoop 专有规则进行配置。

`hadoop.security.auth_to_local` 的值被设为一个或多个从主体名到本地用户名的映射规则。规则可以是以 `DEFAULT` 或 `RULE:` 开头的字符串，其后跟着三部分内容：初始主体转换，接收过滤器、代换命令。特殊值 `DEFAULT` 只映射 Hadoop 本地域名称中的第一部分（如 `alice/admin@EXAMPLE.COM` 被 `DEFAULT` 规则映射为 `alice`）。

1. 初始主体转换

初始主体转换由一个数值加代换字符串组成。该数值对应除域之外的主体成员数量。代换字符串则定义主体如何被初始地转换。变量 `$0` 将指代主体的域，`$1` 将指代第一个成员，`$2` 将指代第二个成员。关于初始主体转换的一些示例见表 5-1。初始主体转换的格式为 **[< *数值* >:< *字符串* >]**，输出称为“*初始本地名称*”。

表5-1：初始主体转换示例

主体转换	alice@EXAM PLE.com的初始本地名称	hdfs/namenode.example.com@EXAMPLE.COM的初始本地名称
`[1:$1.$0]`	`alice.EXAMPLE.COM`	不匹配
`[1: $1]`	`alice`	不匹配
`[2:$1_$2@$0]`	不匹配	`hdfs_namenode.example.com@EXAMPLE.COM`
`[2:$1@$0]`	不匹配	`hdfd@EXAMPLE.COM`

2. 接收过滤器

接收过滤器是个正则表达式。如果初始的本地名称（如规则的第一部分，即初始主体转换的输出）与正则表达式相匹配，那么代换命令则会在该字符串上执行。当且仅当整个字符串与正则表达式匹配时，才会认为初始本地名称符合正则表达式。这相当于正则表达式以 `^` 开始，以 `$` 结束。关于接收过滤器的一些示例见表 5-2。接收过滤器的格式为 **(< *正则表达式* >)**。

表5-2：接收过滤器示例

接收过滤器	alice.EXAMPLE.COM	hdfs@EXAMPLE.COM
`(.*\.EXAMPLE\.COM)`	匹配	不匹配
`(.*@EXAMPLE\.COM)`	不匹配	匹配
`(.*EXAMPLE\.COM)`	匹配	匹配
`(EXAMPLE\.COM)`	不匹配	不匹配

3. 代换命令

代换命令是一个类似 sed 的批量替换命令，该命令由正则表达式和替换字符串组成。可以将一部分正则表达式用括号括起来表示匹配组，并在替换字符串中使用数字（例如 \1）对其进行引用。组号是由左括号在正则表达式中的顺序决定的。表 5-3 是代换命令的一些示例。代换命令的格式为 s/<模式>/<替换字符串>/g。其中，末位的 g 是可选的，若 g 存在，则表示对于整个字符串来说是全局代换；若 g 不存在，则表示只有匹配规则的第一个子串被代换。

表5-3：代换命令示例

代换命令	alice.EXAMPLE.COM	hdfs@EXAMPLE.COM
s/(.*)\.EXAMPLE.COM/\1/	alice	不可用
s/(.EXAMPLE.COM//	alice	hdfs
s/E/Q/	alice.QXAMPLE.COM	hdfs@QXAMPLE.COM
s/E/Q/g	alice.QXAMPLQ.COM	hdfs@QXAMPLQ.COM

一条规则的完整格式为：RULE:[<数字>:<字符串>](<正则表达式>)s/<模式>/<替换字符串>/。多条规则会分行隔开，且会按顺序执行。一旦某一主体完全匹配某一规则（如主体符合初始主体转换的数值，且初始本地名称符合接收过滤器），该规则就会输出该主体的用户名，并且不会继续执行其他规则。由于这样的顺序限制，我们通常将 DEFAULT 规则列在最后。

auth_to_local 设置最常见的使用方式是，用于配置如何处理来自其他 Kerberos 域的主体。一个常见的使用场景是，设置一个或多个信任域。例如，用户的 Hadoop 域为 HADOOP.EXAMPLE.COM，但合作域为 CORP.EXAMPLE.COM，那么用户应添加能将合作域主体转换为本地用户的规则。示例 5-1 给出了仅接受来自 HADOOP.EXAMPLE.COM 域和 CORP.EXAMPLE.COM 域的用户，并且将这两个域的用户映射为两个域首部的配置样例。

示例 5-1　使用 auth_to_local 配置信任域

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
    RULE:[1:$1@$0](.*@CORP.EXAMPLE.COM)s/@CORP.EXAMPLE.COM//
    RULE:[2:$1@$0](.*@CORP.EXAMPLE.COM)s/@CORP.EXAMPLE.COM//
    DEFAULT
  </value>
</property>
```

5.1.2　Hadoop用户到组的映射

Hadoop 使用一个名为 hadoop.security.group.mapping 的配置参数，以控制用户到用户组的映射。默认的实现方法使用标准 UNIX 接口进行本地调用，或者使用 shell 命令查找用户到组的映射。这意味着，只有配置在调用映射的那台服务器上的用户组才对 Hadoop 可见。在实际操作中，这不是个大问题，因为 Hadoop 集群对访问集群的用户和用户组会有一致的视图。



除了要了解用户到组的映射机制如何工作之外，了解这些映射的发生位置也同样重要。正如第 6 章将讲到的，用户到组的映射要始终保持一致，并且要在进行授权决策的环节发生。对于 Hadoop，这意味着映射发生在 NameNode、JobTracker（对 MR1）以及 ResourceManager（对 YARN/MR2）进程中。这是个很重要的细节，因为默认的用户到组的映射实现是通过使用标准 UNIX 接口决定组成员的；对于存在于 Hadoop 视角中的用户组，也必须存在于运行有 NameNode、JobTracker 和 ResourceManager 的服务器上。

`hadoop.security.group.mapping` 的配置参数能够设置于任意实现了 `org.apache.hadoop.security.GroupMappingServiceProvider` 接口的 Java 类中。除了之前介绍过的默认配置，Hadoop 还提供了很多对于该接口的实现，总结如下。

`JniBasedUnixGroupsMapping`
: 一个基于 JNI（Java Native Interface，Java 本地接口）的实现，调用 libc 库函数 `getpwnam_r()` 和 `getgrouplist()` 确定组成员。

`JniBasedUnixGroupsNetgroupMapping`
: `JniBasedUnixGroupsMapping` 的一种扩展，通过调用库函数 `setnetgrent()`、`getnetgrent()` 和 `endnetgrent()` 确定网络组成员。只有服务分层授权访问控制列表中使用到的网络组才会被引入这个映射。

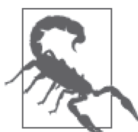
`ShellBasedUnixGroupsMapping`
: 一种基于 shell 的实现，使用 `id -Gn` 命令。

`ShellBasedUnixGroupsNetgroupMapping`
: `ShellBasedUnixGroupsMapping` 的一种扩展，使用 `getent netgroup shell` 命令确定网络组成员。只有服务分层授权访问控制列表中使用到的网络组才会被引入这个映射。

`JniBasedUnixGroupsMappingWithFallback`
: `JniBasedUnixGroupsMapping` 类的封装，若不能加载本地库（这是默认设置下的实现），则会回退到 `ShellBasedUnixGroupsMapping` 类。

`JniBasedUnixGroupsNetgroupMappingWithFallback`
: `JniBasedUnixGroupsNetgroupMapping` 类的封装，若不能加载本地库，则会回退到 `ShellBasedUnixGroupsNetgroupMapping` 类。

`LdapGroupsMapping`
: 与 LDAP 或 Active Directory 服务器直连，确定组成员。



无论组映射是如何配置的，Hadoop 都会将映射内容缓存，并且仅在缓存记录过期后才调用映射的实现。组映射缓存默认为每隔 300 秒（5 分钟）过期。若用户想使底层组的更新更频繁地在 Hadoop 中体现，则要将 *core-site.xml* 中的 `hadoop.security.groups.cache.secs` 设置为希望记录被缓存的秒数。这个数值应设得尽量小，以迅速反映更新的变化。然而又不能太小，否则将导致对 LDAP 服务器或其他用户组提供者的不必要的访问。

使用LDAP将用户映射至组

大多数应用部署可以使用默认的组映射来源。然而，对于仅能直接从 LDAP 或 Active Directory 服务器——而不是从集群节点——引用群组的环境，Hadoop 提供了 `LdapGroupsMapping` 实现。该方法的配置通过设置位于 NameNode、JobTracker 和 / 或 ResourceManager 上的 `core-site.xml` 文件中的数个参数实现。

`hadoop.security.group.mapping.ldap.url`
: 用于解析用户群组的 LDAP 服务器 URL。必须以 `ldap://` 或者 `ldaps://`（若 SSL 功能开启）开头。

`hadoop.security.group.mapping.ldap.bind.user`
: 连接 LDAP 服务器时给用户绑定的唯一名称。该用户需要对目录进行读访问，并且不能是管理员。

`hadoop.security.group.mapping.ldap.bind.password`
: 已绑定用户的密码。最好不要使用该参数设置密码，而将密码放进一个独立的文件，并且将 `hadoop.security.group.mapping.ldap.bind.password.file` 指向该独立文件的路径。



如果将 Hadoop 配置为直接使用 LDAP，则会缺失 Hadoop 服务账户（如 hdfs）的本地组。这样会导致 log 日志中产生大量如下消息：

```
No groups available for user hdfs
```

因此，使用 JNI 或基于 shell 的映射，并在操作系统层将 LDAP/Active Directory 集成的做法通常更好。系统安全服务守护进程（SSDD）融合了许多身份和验证系统，并且能为缓存和离线访问提供通用支持。

使用上述参数，示例 5-2 展示了如何在 `coresite.xml` 中实现 `Ldap Groups Mapping`。

示例 5-2　core-site.xml 中的 LDAP 映射示例

```
...
<property>
  <name>hadoop.security.group.mapping</name>
  <value>org.apache.hadoop.security.LdapGroupsMapping</value>
</property>
<property>
  <name>hadoop.security.group.mapping.ldap.url</name>
  <value>ldap://ad.example.com</value>
</property>
<property>
  <name>hadoop.security.group.mapping.ldap.bind.user</name>
  <value>Hadoop@ad.example.com</value>
</property>
<property>
  <name>hadoop.security.group.mapping.ldap.bind.password</name>
  <value>password</value>
</property>
...
```

除了必需的参数外，还有几个可选参数能够控制用户和组之间是如何映射的。

`hadoop.security.group.mapping.ldap.bind.password.file`
: 本参数为含有绑定用户密码的文件路径。该文件应该只能被运行服务（通常是 `hdfs`、`mapred` 和 `yarn`）的 UNIX 用户读取。

`hadoop.security.group.mapping.ldap.ssl`
: 本参数值为 true 时，开启连接 LDAP 服务器的 SSL 协议。若该设定为生效状态，则 `hadoop.security.group.mapping.ldap.url` 参数必须以 `ldaps://` 起始。

`hadoop.security.group.mapping.ldap.ssl.keystore`
: 本参数为 Java 密钥库的路径，包含开启 SSL 时 LDAP 服务器要求的客户证书。该密钥库必须为 JKS（Java 密钥库）格式。

`hadoop.security.group.mapping.ldap.ssl.keystore.password`
: 本参数为 `hadoop.security.group.mapping.ldap.ssl.keystore` 文件的密码。最好不要使用该设置，而将密码放在另一个独立的文件中，并将 `hadoop.security.group.mapping.ldap.ssl.keystore.password.file` 属性配置为指向该独立文件的路径。

`hadoop.security.group.mapping.ldap.ssl.keystore.password.file`
: 本参数为包含 `hadoop.security.group.mapping.ldap.ssl.keystore` 文件密码的文件路径。该文件应该只能被运行服务（通常是 `hdfs`、`mapred` 和 `yarn`）的 UNIX 用户读取。

`hadoop.security.group.mapping.ldap.base`
: 该参数为用于搜索 LDAP 目录的搜索库。搜索库的名称唯一，且通常会在能够覆盖访问集群的所有用户的前提下被配置得尽可能具体。

`hadoop.security.group.mapping.ldap.search.filter.user`
: 用于搜索 LDAP 用户的过滤器。默认配置是 `(&(objectClass=user)(sAMAccountName={0}))`，通常适用于安装了 Active Directory 的情况。对于其他 LDAP 服务器，需要修改该配置。对于 OpenLDAP 和相应的兼容服务器，推荐的配置为 `(&(objectClass=inetOrgPerson)(uid={0}))`。

`hadoop.security.group.mapping.ldap.search.filter.group`
: 用于搜索 LDAP 用户组的过滤器。默认配置为 `(objectClass=group)`，通常适用于安装了 Active Directory 的情况。

`hadoop.security.group.mapping.ldap.search.attr.member`
: 组对象用于标识组内用户身份的属性。

`hadoop.security.group.mapping.ldap.search.attr.group.name`
: 组对象用于标识组名的属性。

`hadoop.security.group.mapping.ldap.directory.search.timeout`
: 等待搜索目录结果的超时时间，以毫秒为单位。

5.1.3 Hadoop用户配置

最难理解的 Hadoop 安全需求之一就是，集群的所有用户都必须能在集群中的所有服务器上进行配置。这意味着他们既可以存在于本地的 /etc/passwd 文件中，也可以通过服务器接入的网络目录服务进行配置（这种情况更普遍），例如 OpenLDAP 或者 Active Directory。为了理解该需求，需要牢记，Hadoop 实际上是一个允许用户在集群机器上提交和执行任意代码的服务。这意味着，如果不信任用户，则需要限制这些用户对任何或所有运行在集群服务器上的服务进行访问，包括本地文件系统等标准 Linux 服务。目前，加强这些限制的最好方法是，在集群上使用提交任务的用户的用户名和 UID 执行各项任务。为了满足此需求，必须让集群中的每一台服务器都使用一致的用户数据库。



虽然所有集群用户都必须在集群上的所有服务器上进行配置，但不一定要给所有用户都开启本地或远程 shell 访问。最好的策略是，通过预设的 `shell/sbin/nologin` 对用户进行配置，并且通过 `/etc/ssh/sshd_config` 文件中的 `AllowUsers`、`DenyUsers`、`AllowGroups` 和 `DenyGroups` 配置禁止用户的 SSH 访问。

5.2 身份验证

Hadoop 的早期版本以及相关生态系统的项目是不支持强认证的。Hadoop 是个复杂的分布式系统，幸运的是，它的生态系统中，绝大多数组件都根据有限的几个认证方式进行了标准化，这些标准化取决于具体的服务和协议。特别地，Kerberos 被用于生态系统中的绝大部分组件内，因为 Hadoop 在早期的安全属性开发时就已经根据它进行了标准化。表 5-4 是根据服务和协议总结的身份验证方法。本节重点关注 HDFS、MapReduce、YARN、HBase、Accumulo 和 ZooKeeper 的身份验证。Hive、Impala、Hue、Oozie 和 Solr 的身份验证将延后至第 11 章和第 12 章进行介绍，因为这些通常是由用户直接访问的。

表5-4：Hadoop生态系统的身份验证方法

服务	协议	方法
HDFS	RPC	Kerberos，委托令牌
HDFS	Web UI	SPNEGO（Kerberos），可插拔式
HDFS	REST(WebHDFS)	SPNEGO（Kerberos），委托令牌
HDFS	REST(HttpFS)	SPNEGO（Kerberos），委托令牌
MapReduce	RPC	Kerberos，委托令牌
MapReduce	Web UI	SPNEGO（Kerberos），可插拔式
YARN	RPC	Kerberos，委托令牌
YARN	Web UI	SPNEGO（Kerberos），可插拔式
Hive Server 2	Thrift	Kerberos, LDAP（用户名 / 密码）
Hive Metastore	Thrift	Kerberos, LDAP（用户名 / 密码）
Impala	Thrift	Kerberos, LDAP（用户名 / 密码）

（续）

服务	协议	方法
HBase	RPC	Kerberos，委托令牌
HBase	Thrift Proxy	无
HBase	REST Proxy	SPNEGO（Kerberos）
Accumulo	RPC	用户名 / 密码，可插拔式
Accumulo	Thrift Proxy	用户名 / 密码，可插拔式
Solr	HTTP	基于 HTTP 容器
Oozie	REST	SPNEGO（Kerberos, 委托令牌）
Hue	Web UI	用户名 / 密码（数据库、PAM、LDAP）、SAML、OAuth、SPNEGO（Kerberos）、远程用户（HTTP 代理）
ZooKeeper	RPC	Digest（用户名 / 密码）、IP、SASL（Kerberos）、可插拔式

5.2.1 Kerberos

Hadoop 支持两种身份验证机制：简单机制和 Kerberos 机制。简单机制是默认设置，根据客户进程的有效 UID 确定用户名，该用户名会不附带任何额外凭证，直接传给 Hadoop。这种模式下，Hadoop 服务完全信任他们的客户。这种默认设置允许任何可访问集群的用户均能被完全信任，直接访问上述集群的所有数据和管理员级功能。对于概念验证性系统或实验环境，通常会允许系统以这种模式运行，并使用防火墙对其进行保护，同时会限制那些能够使用客户端访问集群内任何系统的用户集。然而对于产品级或者多租户的系统，这样的身份验证机制几乎不能被接受。简单认证同样被 HBase 作为其默认的身份验证机制。

HDFS、MapReduce、YARN、HBase、Oozie 和 ZooKeeper 都支持 Kerberos 作为客户端的身份验证机制，虽然在实现方面会由于服务和接口的区别而有些不同。对基于 RPC 的协议，简单认证与安全层（**Simple Authentication and Security Layer**，**SASL**）框架被用于向下层协议添加认证。理论上，任意 SASL 协议都能得到支持；但实际上，真正仅支持的机制是 GSSAPI（Kerberos V5）和 DIGEST-MD5（DIGEST-MD5 的详细内容参见 5.2.3 节）。Oozie 没有 RPC 协议，取而代之的是，它为用户提供了 REST 接口。Oozie 使用简单和受保护的 **GSSAPI** 协商机制（**Simple and Protected GSSAPI Negotiation Mechanism**，**SPNEGO**），该协议最初由微软在 Internet Explorer 5.0.1 和 IIS 5.0 中为 HTTP 层的 Kerberos 认证实现。HDFS、MapReduce、YARN、Oozie 和 Hue 的 Web 接口，以及 HDFS（包括 WebHDFS 和 HttpFS）和 HBase 的 REST 接口也同样支持 SPNEGO。SASL 和 SPNEGO 的身份验证过程遵循标准的 Kerberos 协议，只有提交服务票据的机制发生改变。

让我们来看 Alice 是如何通过 Kerberos 与 HDFS 的 NameNode 进行验证交互的：

- Alice 为由 `hdfs/namenode.example.com@EXAMPLE.COM` 标识的 HDFS 服务，从 `kdc.example.com` 的 TGS 请求一个服务票据，并随请求附上她的 TGT。
- TGS 验证 TGT 之后，提供给 Alice 一个服务票据，使用 `hdfs/namenode.example.com@EXAMPLE.COM` 的主体密钥。

- Alice 把服务票据提交给 NameNode（通过 SASL），NameNode 能够使用 `hdfs/namenode.example.com@EXAMPLE.COM` 的密钥对票据进行解密和认证。

5.2.2 用户名和密码验证

ZooKeeper 支持基于用户名和密码的认证。相比于使用一个用户名和密码的数据库，ZooKeeper 将验证密码的环节推迟到授权步骤（详见 6.4 节 ZooKeeper ACLs）。一个 ACL（Access Control List，访问控制列表）被附加到一个 ZNode 上时，会引入验证方案和方案 ID。这个 ID 是该方案的认证提供者验证过的。用户名和密码的认证通过摘要式身份验证提供者实现，它会生成用户名和密码的一个 SHA-1 摘要值。由于认证环节被推迟到授权步骤，因此认证步骤总能通过。用户通过调用 `addAuthInfo(String scheme, byte[] authData)` 方法添加他们的验证细节，其中 `<username>:<password>.getBytes()` 作为鉴别数据时，`<username>` 和 `<password>` 会被相应的值替换。

Accumulo 也支持基于用户名和密码的验证。与 ZooKeeper 不同，Accumulo 使用更常见的方法存储用户名和密码，并且有一个清晰明确的登录步骤验证密码有效性。Accumulo 的验证体系对 `AuthenticationToken` 接口进行不同的实现，以保证其可插拔性。最常见的一种实现是 `PasswordToken` 类，该类可以从 `CharSequence` 或 Java 属性文件初始化。使用用户名和密码连接 Accumulo 的样例代码如示例 5-3 所示。

示例 5-3 使用用户名和密码连接 Accumulo

```
//创建一个Accumulo实例的句柄
Instance instance = new ZooKeeperInstance("instance",
    "zk1.example.com,zk2.example.com,zk3.example.com");

// 创建一个包含密码的令牌
AuthenticationToken token = new PasswordToken("secret");

//创建连接器;如果密码无效,则会抛出AccumuloSecurityException异常
Connector connector = instance.getConnector("alice", token);
```

5.2.3 令牌

任何分布式系统内，以用户名义执行的所有行为都有必要验证该用户身份。仅仅验证服务的归属者是不够的，验证必须发生在每一次交互中。以运行一个 MapReduce 作业为例，想要扩展命令行参数中的任意通配符，需要在客户端和 NameNode 之间进行身份验证；为了提交作业，则需要在客户端和 JobTracker 之间均进行身份验证。

JobTracker 将工作分解为被每个集群中的 TaskTracker 顺序执行的任务，每个任务必须与 NameNode 通信，以打开组成输入分片（input split）的文件。为了加强文件系统的许可控制，每个任务必须通过 NameNode 的身份验证。若 Kerberos 是唯一的身份验证机制，用户的 TGT 需要分派给每一个任务。该方法的缺点在于，这样做会允许任务向任意一个受 Kerberos 保护的服务发起身份验证请求，而这种做法是不合适的。通过签发能分派给每个任务、但被限制在某个特定服务内的认证令牌，Hadoop 解决了该问题。

1. 委托令牌

Hadoop 有多种令牌，用于在没有 TGT 或 Kerberos 服务票据的情况下进行认证后的访问。通过 Kerberos 与 NameNode 完成身份验证后，一个客户端能够获得一个*委托令牌*。委托令牌其实是客户端和 NameNode 之间共享的密钥，能够通过 DIGEST-MD5 机制用于 RPC 身份验证。

图 5-1 展示了客户端和 NameNode 之间的两次交互。首先，客户端使用 Kerberos 服务票据进行了 `getDelegationToken()` 的 RPC 调用，以请求一个委托令牌 (1)。NameNode 回复了一个委托令牌 (2)。客户端发起 `getListing()` 的 RPC 调用，以请求目录列表，但这次客户端使用的是供身份验证用的委托令牌。验证令牌后，NameNode 会给客户端响应，内容为 `DirectoryListing`(4)。

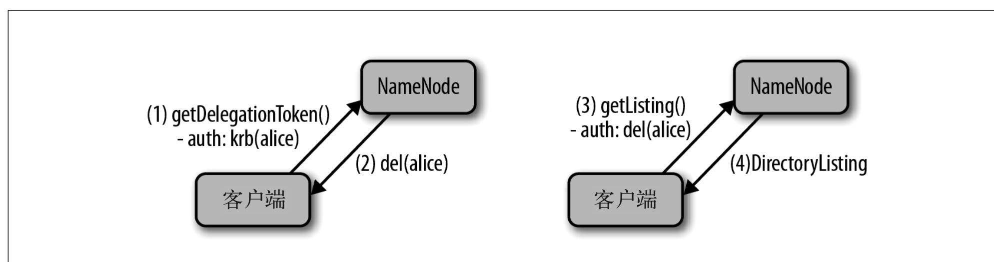


图 5-1：获取并使用委托令牌

令牌既有失效日期，也有最大发行日期。令牌在失效日期之后会失效，但直至其最大发行日期前，仍然能够被更新。客户端能够在完成向 NameNode 的初始 Kerberos 认证之后，发起对委托令牌的请求。令牌会有一个指定的*更新机制*。为用户更新令牌时，令牌更新器使用其 Kerberos 凭证进行认证。委托令牌最常见的用法是在 MapReduce 任务中，客户端指定 JobTracker 为令牌更新器。委托令牌由 NameNode 的 URL 生成，且存储在 JobTracker 的系统目录下，以便将其传给任务。该方法使得任务访问 HDFS 时，不会将用户的 TGT 置于泄露风险中。

2. 块访问令牌

文件权限检查由 NameNode 执行，而非 DataNode。默认情况下，任何客户端只要提供块 ID 就可以访问任意块。为了解决这个问题，Hadoop 引入了*块访问令牌*的概念。块访问令牌由 NameNode 生成，在客户端通过身份验证，且 NameNode 已执行完访问文件 / 块的必要授权检查后，会将其发送给客户端。块访问令牌包括客户端 ID、块 ID 和允许访问模式（READ、WRITE、COPY、REPLACE），并且使用 NameNode 和 DataNode 之间的共享密钥进行签名。该共享密钥永远不会让客户端获得，且块访问令牌过期时，客户端必须向 NameNode 重新请求一个令牌。

图 5-2 展示了客户端如何使用块访问令牌读取文件。首先，客户端使用 Kerberos 凭证，通过 `getBlockLocations()` 的 RPC 调用，向 NameNode 请求所需的块位置 (1)。NameNode 会响应一个 `LocatedBlock` 对象，该对象包括针对被请求块的块访问令牌 (2)。客户端使用块访问令牌，通过数据传输协议中的 `readBlock()` 方法，向 DataNode 进行身份验证并请求数

据 (3)。最后，DataNode 将响应请求的数据 (4)。

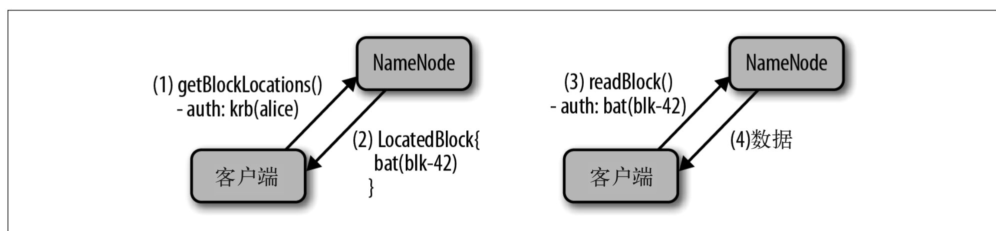


图 5-2：使用块访问令牌访问块

3. 作业令牌

提交一个 MapReduce 作业时，JobTracker 会创建一个名为“作业令牌”的密钥，该令牌被作业中的任务用于向 TaskTracker 发起身份验证。JobTracker 将作业令牌放置在 HDFS 中 JobTracker 的系统目录下，并通过 RPC 分发给各个 TaskTracker。TaskTracker 会将该令牌存放在本地磁盘的作业目录下，该目录仅能被作业用户访问。作业令牌用于给任务和 TaskTracker 之间的 RPC 通信进行身份验证，也用于生成一种散列值，该散列值确保通过 HTTP 传输的中间输出在 shuffle 阶段仅能被该作业的任务访问。此外，返回 shuffle 数据的 TaskTracker 会计算一个散列值，供每个任务验证它是在同真实的 TaskTracker——而不是伪装者——进行对话。

图 5-3 的时序图展示了作业建立阶段使用的身份验证方法。首先，客户端通过 Kerberos 认证请求建立一个新作业 (1)。JobTracker 给客户端响应作业 ID，用于该作业的唯一标识 (2)。客户端随之向 NameNode 请求委托令牌，并使用 JobTracker 作为更新器 (3)。NameNode 给客户端响应委托令牌 (4)。只有客户端通过 Kerberos 身份验证后，才会被分发委托令牌。最后，通过 Kerberos 与 JobTracker 进行身份验证的客户端将委托令牌与其他所需的作业细节发送给 JobTracker。

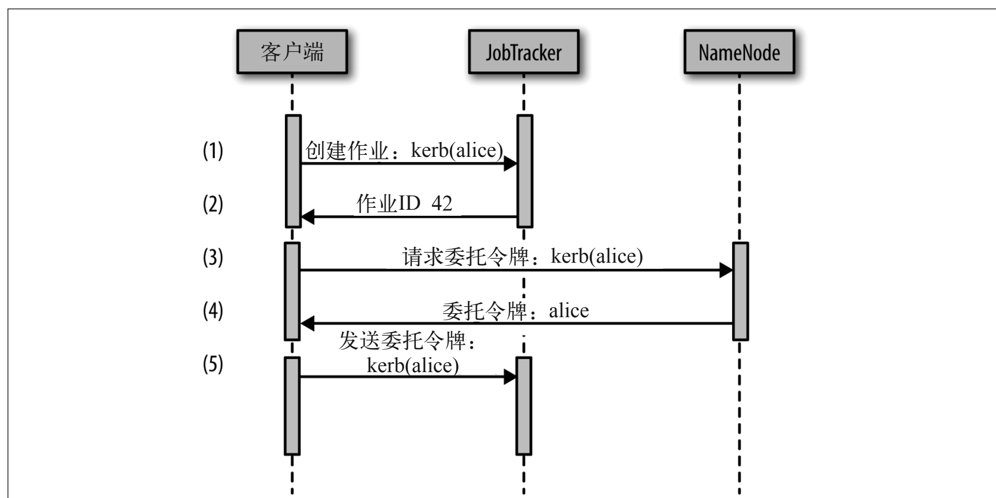


图 5-3：建立作业

开始执行作业时，情况会更加有意思，如图 5-4 所示。

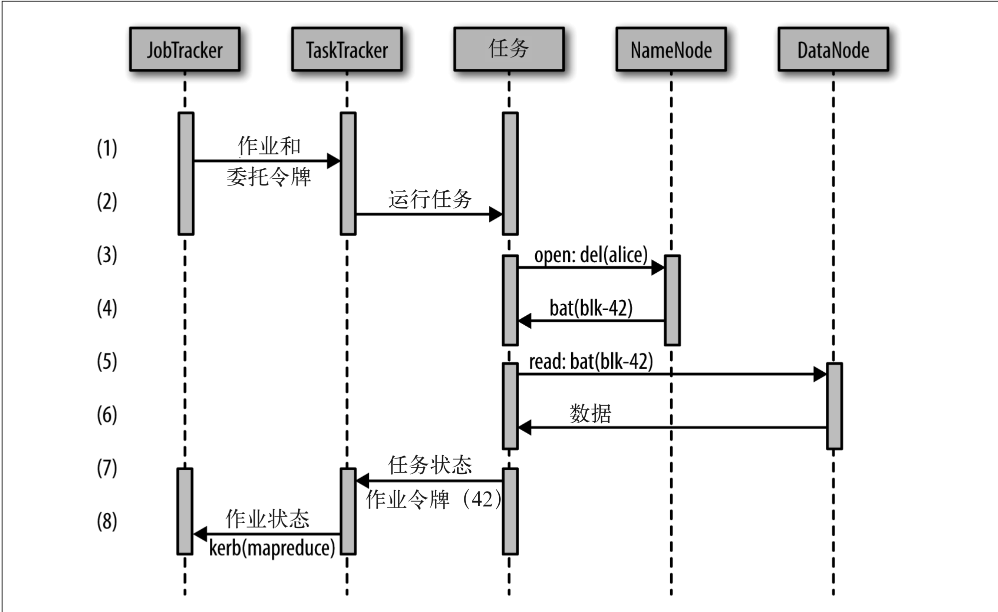


图 5-4：执行作业

JobTracker 会生成一个作业令牌，并将作业令牌、委托令牌和其他所需信息打包发送给 TaskTracker(1)。JobTracker 与 TaskTracker 对话时，需经过 Kerberos 身份验证。TaskTracker 会把接收到的令牌存储至一个仅能被提交作业的用户访问的目录，然后运行任务 (2)。任务通过委托令牌打开文件，并请求块地址以获得输入分片 (3)。NameNode 会将块地址和相应的块访问令牌响应给发起请求的任务 (4)。该任务随之使用获得的块访问令牌，从 DataNode 请求读取数据 (5)，DataNode 随之给予响应 (6)。作业执行过程中，任务会通过作业令牌进行身份验证，向 TaskTracker 报告任务状态 (7)。然后 TaskTracker 会通过 Kerberos 身份验证，并向 JobTracker 报告状态，从而收集整体作业状态 (8)。

5.2.4 用户模拟

Hadoop 生态系统中，有许多服务能代表终端用户执行事务。为了保证安全，这些服务必须对自己的客户端进行身份验证，并且能够被信任，以模拟其他用户。Oozie、Hive（HiveServer2 中）和 Hue 均支持，访问 HDFS、MapReduce、YARN 和 HBase 时模拟终端用户。安全模拟工作贯穿于这些服务，它们指定哪些用户可以执行模拟，以这种方式支持安全模拟。一个受信任的用户需要代表另一个用户执行操作时，她必须验证自己的身份，并提供其代替用户的用户名。受信任用户可以被限制为仅能模拟特定用户组，且为了进一步约束这些用户的特权，还可以限制他们仅能从某些主机访问 Hadoop。

模拟有时也称*代理机制*。能够执行模拟行为的用户（如能代表其他用户的）称为*代理*。启用模拟的配置参数为 `hadoop.proxyuser.<proxy>.hosts` 和 `hadoop.proxyuser.<proxy>.`

groups，其中 <proxy> 是执行模拟行为用户的用户名。参数值为按逗号分隔、且顺序排列的主机和组列表，若为 * 则意味着所有主机 / 组。若用户同时需要 Hue 和 Oozie 的代理功能，但又需要限制 Oozie 能够代理的 oozie-user 组的成员，则需要使用示例 5-4 中的配置。

示例 5-4　代理的示例配置

```
<!-- Configure Hue impersonation from hue.example.com -->
<property>
  <name>hadoop.proxyuser.hue.hosts</name>
  <value>hue.example.com</value>
</property>
<property>
  <name>hadoop.proxyuser.hue.groups</name>
  <value>*</value>
</property>

<!--
    Configure Oozie impersonation from oozie01.example.com and
    oozie02.example.com for users in oozie-users
  -->
<property>
  <name>hadoop.proxyuser.oozie.hosts</name>
  <value>oozie01.example.com,oozie02.example.com</value>
</property>
<property>
  <name>hadoop.proxyuser.oozie.groups</name>
  <value>oozie-users</value>
</property>
```

5.2.5　配置

对于产品部署，Hadoop 支持使用 Kerberos 机制进行身份验证。按照 Kerberos 的身份验证机制配置时，所有用户和守护进程必须提供有效的凭证以访问 PRC 接口。这意味着，必须替集群中的每一个服务 / 守护进程对都创建一个 Kerberos 服务主体。回顾第 4 章讲过的服务主体名称（SPN）的概念，它由三部分组成：服务名称、主机名称、域。Hadoop 中，每个隶属于某服务的守护进程都会使用该服务的名称（HDFS 服务用 hdfs、MapReduce 服务用 mapred、YARN 服务用 yarn）。另外，若要为多个 Web 接口启用 Kerberos 身份验证机制，还需要向主体提供 HTTP 服务的名称。

下面了解如何配置一个使用 Kerberos 身份验证机制的集群。假设有一个集群，包含表 5-5 所示的主机和服务。



表 5-5 展示的服务层仅用作示例，并不是配置集群的最好选择。对于初学者，我们使用 YARN 和 MR1 服务的配置展示示例。这仅意味着，给出全方位的配置示例需要用到这两个服务（并不是说用户必须使用这两个服务）。实际部署中，可能只部署一个或另一个。与之类似，如果像在本例中使用 HA 运行两个 NameNode，也不需要再另外部署一个 SecondaryNameNode。重申一下，这仅仅是为了让我们的示例更全面而详尽。

表5-5：服务层

主机名	守护进程
nn1.example.com	NameNode
	JournalNode
nn2.example.com	NameNode
	JournalNode
snn.example.com	SecondaryNameNode
	JournalNode
rm.example.com	ResourceManager
jt.example.com	JobTracker
	JobHistoryServer
dn1.example.com	DataNode
	TaskTracker
	NodeManager
dn2.example.com	DataNode
	TaskTracker
	NodeManager
dn3.example.com	DataNode
	TaskTracker
	NodeManager

第一步，要在 Kerberos KDC 中创建所需的 SPN，并为每台服务器上的每一个守护进程导出 keytab（密钥库）文件。每个主机 / 角色所需 SPN、它们各自所属的 keytab 文件的推荐名称如表 5-6 所示。对于每个服务器需要创建不同的 keytab 文件。为了能在所有主机上使用相同配置（此时不同服务器上的同名 keytab 文件含有不同密钥），建议每个守护进程都使用一致的名称。

表5-6：所需Kerberos规则

主机名	守护进程	密钥文件	SPN
nn1.example.com	NameNode/JournalNode	hdfs.keytab	hdfs/nn1.example.com@EXAMPLE.COM
			HTTP/nn1.example.com@EXAMPLE.COM
nn2.example.com	NameNode/JournalNode	hdfs.heytab	hdfs/nn2.example.com@EXAMPLE.COM
			HTTP/nn2.example.com@EXAMPLE.COM
snn.example.com	SecondaryNameNode/JournalNode	hdfs.keytab	hdfs/snn.example.com@EXAMPLE.COM
			HTTP/snn.example.com@EXAMPLE.COM
rm.example.com	ResourceManager	yarn.keytab	yarn/rm.example.com@EXAMPLE.COM
jt.example.com	JobTracker	mapred.keytab	mapred/jt.example.com@EXAMPLE.COM
			HTTP/jt.example.com@EXAMPLE.COM
	JobHistoryServer	mapred.keytab	mapred/jt.example.com@EXAMPLE.COM

（续）

主机名	守护进程	密钥文件	SPN
dn1.example.com	DataNode	hdfs.keytab	hdfs/dn1.example.com@EXAMPLE.COM
			HTTP/dn1.example.com@EXAMPLE.COM
	TaskTracker	mapred.keytab	mapred/dn1.example.com@EXAMPLE.COM
			HTTP/dn1.example.com@EXAMPLE.COM
	NodeManager	yarn.keytab	yarn/dn1.example.com@EXAMPLE.COM
			HTTP/dn1.example.com@EXAMPLE.COM
dn2.example.com	DataNode	hdfs.keytab	hdfs/dn2.example.com@EXAMPLE.COM
			HTTP/dn2.example.com@EXAMPLE.COM
	TaskTracker	mapred.keytab	mapred/dn2.example.com@EXAMPLE.COM
			HTTP/dn2.example.com@EXAMPLE.COM
	NodeManager	yarn.keytab	yarn/dn2.example.com@EXAMPLE.COM
			HTTP/dn2.example.com@EXAMPLE.COM
dn3.example.com	DataNode	hdfs.keytab	hdfs/dn3.example.com@EXAMPLE.COM
			HTTP/dn3.example.com@EXAMPLE.COM
	TaskTracker	mapred.keytab	mapred/dn3.example.com@EXAMPLE.COM
			HTTP/dn3.example.com@EXAMPLE.COM
	NodeManager	yarn.keytab	yarn/dn3.example.com@EXAMPLE.COM
			HTTP/dn3.example.com@EXAMPLE.COM



导出 keytab 文件时需要注意，因为默认的设置是，每导出一个主体就将 Kerberos 密钥随机化。用户可以将每个主体导出，然后使用 `ktutil` 工具将每个守护进程的密钥和 keytab 文件结合起来。

我们推荐将相应的 keytab 文件存储在 `$HADOOP_CONF_DIR` 目录中（一般是 `/etc/hadoop/conf`）。



配置文件完整示例

配置文件的完整示例在本书 GitHub 主页的示例资源（http://bit.ly/1EAEmTr）中。

创建所需的 SPN 并分发 keytab 后，需要对 Hadoop 进行配置，以使用 Kerberos 进行身份验证。如示例 5-5 所示，首先将 `core-site.xml` 文件的 `hadoop.security.authentication` 参数设为 `kerberos`。

示例 5-5　将身份验证类型配置为 Kerberos

```
<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value>
</property>
```

1. HDFS

接下来，需要为每个服务配置 Kerberos 主体和 keytab 文件。对于 NameNode，还必须设置 `dfs.block.access.token.enable` 为 `true`，以启用块访问令牌。NameNode 的配置应当在 `hdfs-site.xml` 文件中进行，如示例 5-6 所示。

示例 5-6　配置 NameNode

```
<property>
  <name>dfs.block.access.token.enable</name>
  <value>true</value>
</property>
<property>
  <name>dfs.namenode.keytab.file</name>
  <value>hdfs.keytab</value>
</property>
<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>hdfs/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>dfs.namenode.kerberos.internal.spnego.principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
```

如果没有启用 HDFS 的高可用模式，接下来则需要在 `hdfs-site.xml` 文件中对 SecondaryNameNode 进行配置，如示例 5-7 所示。

示例 5-7　配置 SecondaryNameNode

```
<property>
  <name>dfs.secondary.namenode.keytab.file</name>
  <value>hdfs.keytab</value>
</property>
<property>
  <name>dfs.secondary.namenode.kerberos.principal</name>
  <value>hdfs/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>dfs.secondary.namenode.kerberos.internal.spnego.principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
```

如果启用了 HDFS 的高可用模式，则需要在 `hdfs-site.xml` 文件中对 JournalNode 进行如下配置，如示例 5-8 所示。

示例 5-8　配置 JournalNode

```
<property>
  <name>dfs.journalnode.keytab.file</name>
  <value>hdfs.keytab</value>
</property>
<property>
  <name>dfs.journalnode.kerberos.principal</name>
```

```
    <value>hdfs/_HOST@EXAMPLE.COM</value>
  </property>
  <property>
    <name>dfs.journalnode.kerberos.internal.spnego.principal</name>
    <value>HTTP/_HOST@EXAMPLE.COM</value>
  </property>
```

接下来，按照如下设置，在 `hdfs-site.xml` 文件中配置 DataNode。除了要配置 keytab 和主体名之外，还必须对 DataNode 进行设置，使其对 RPC 和 HTTP 服务使用特权端口。之所以使用特权端口是因为，DataNode 的数据传输协议并不使用 Hadoop 的 RPC 框架。通过特权端口，DataNode 能够验证其是由 `root` 通过 `jsvc` 发起的，如示例 5-9 所示。

示例 5-9　配置 DataNode

```
  <property>
    <name>dfs.datanode.address</name>
    <value>0.0.0.0:1004</value>
  </property>
  <property>
    <name>dfs.datanode.http.address</name>
    <value>0.0.0.0:1006</value>
  </property>
  <property>
    <name>dfs.datanode.keytab.file</name>
    <value>hdfs.keytab</value>
  </property>
  <property>
    <name>dfs.datanode.kerberos.principal</name>
    <value>hdfs/_HOST@EXAMPLE.COM</value>
  </property>
```

WebHDFS 是基于 REST 的数据访问协议。WebHDFS 的职责是，从存有被读取块的 DataNode 中读取数据，并通过 HTTP 对外提供数据。为了安全访问 WebHDFS，需要设置存于 NameNode 和 DataNode 的 `hdfs-site.xml` 文件中的下列参数，如示例 5-10 所示。

示例 5-10　配置 WebHDFS

```
  <property>
    <name>dfs.web.authentication.kerberos.keytab</name>
    <value>hdfs.keytab</value>
  </property>
  <property>
    <name>dfs.web.authentication.kerberos.principal</name>
    <value>HTTP/_HOST@EXAMPLE.COM</value>
  </property>
```

现在，HDFS 的配置就算完成了！

2. YARN

现在配置 YARN，从 ResourceManager 开始。先设置 `yarn-site.xml` 文件中的配置参数，如示例 5-11 所示。

示例 5-11　配置 ResourceManager

```
<property>
  <name>yarn.resourcemanager.principal</name>
  <value>yarn/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>yarn.resourcemanager.webapp.spnego-principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>yarn.resourcemanager.keytab</name>
  <value>yarn.keytab</value>
</property>
<property>
  <name>yarn.resourcemanager.webapp.spnego-keytab-file</name>
  <value>yarn.keytab</value>
</property>
```

在 `yarn-site.xml` 文件中对参数进行设置，将 NodeManager 配置为使用 Kerberos，如示例 5-12 所示。

示例 5-12　配置 NodeManager

```
<property>
  <name>yarn.nodemanager.principal</name>
  <value>yarn/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>yarn.nodemanager.webapp.spnego-principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>yarn.nodemanager.keytab</name>
  <value>yarn.keytab</value>
</property>
<property>
  <name>yarn.nodemanager.webapp.spnego-keytab-file</name>
  <value>yarn.keytab</value>
</property>
```

除了将 NodeManager 配置为使用 Kerberos 进行身份验证外，还需要设置 NodeManager 使用 `LinuxContainerExecutor`。`LinuxContainerExecutor` 使用一个 setuid 程序启动 YARN 容器，这使得 NodeManager 能够使用提交作业的用户 UID 执行容器。因此，需要一个安全的配置，以确保 Alice 不能访问 Bob 执行的容器所创建的文件。如果没有 `LinuxContainerExecutor`，`yarn` 用户和容器就能够相互访问彼此的本地文件，这会造成所有容器都被运行。首先，配置 `yarn-site.xml` 文件，如示例 5-13 所示。

示例 5-13　配置 NodeManager 使用 LinuxContainerExecutor

```
<property>
  <name>yarn.nodemanager.container-executor.class</name>
  <value>org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor</value>
</property>
```

```
<property>
  <name>yarn.nodemanager.linux-container-executor.group</name>
  <value>yarn</value>
</property>
```

还需要配置执行器程序本身，配置 `container-executor.cfg` 文件中的参数即可，如示例 5-14 所示。`yarn.nodemanager.linux-container-executor.group` 参数的值应当与 `yarn-site.xml` 文件和 `container-executor.cfg` 文件中的值保持一致。通常该值被设为 `yarn`。

示例 5-14　配置 LinuxContainerExecutor

```
yarn.nodemanager.linux-container-executor.group=yarn
min.user.id=1000
allowed.system.users=nobody,impala,hive,llama
banned.users=root,hdfs,yarn,mapred,bin
```

对 `min.user.id` 的设定用于防止 `LinuxContainerExecutor` 运行 UID 小于该参数值的容器。该值通常被设为 1000 或 500，具体取决于普通用户的 UID 在实际环境中的开始位置。除了该设置之外，还可以设置明确的用户白名单和黑名单。这种设置专门允许用户中的 **hive** 用户运行容器。启用 Apache Sentry 时需要进行该项配置，因为 Sentry 启用时，Hive 代理是被关闭的。

配置 YARN 使用 Kerberos 的最后一步是配置 JobHistoryServer，设定 `mapredsite.xml` 文件中的参数即可，如示例 5-15 所示。

示例 5-15　为 Kerberos 配置 JobHistoryServer

```
<property>
  <name>mapreduce.jobhistory.principal</name>
  <value>mapred/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>mapreduce.jobhistory.webapp.spnego-principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>mapreduce.jobhistory.keytab</name>
  <value>mapred.keytab</value>
</property>
<property>
  <name>mapreduce.jobhistory.webapp.spnego-keytab-file</name>
  <value>mapred.keytab</value>
</property>
```

3. MapReduce (MR1)

如果用户仍在使用 MR1，则可以跳过上面的 YARN 配置，直接对 JobTracker 和 TaskTracker 进行配置。首先，配置 `mapred-site.xml` 文件中的参数，如示例 5-16 所示。

示例 5-16　为 Kerberos 配置 JobTracker

```
<property>
  <name>mapreduce.jobtracker.kerberos.principal</name>
  <value>mapred/_HOST@EXAMPLE.COM</value>
```

```
  </property>
  <property>
    <name>mapreduce.jobtracker.keytab.file</name>
    <value>mapred.keytab</value>
  </property>
```

对 TaskTracker 的配置也同样简单。配置 `mapred-site.xml` 文件中的参数，如示例 5-17 所示。

示例 5-17　为 Kerberos 配置 TaskTracker

```
  <property>
    <name>mapreduce.tasktracker.kerberos.principal</name>
    <value>mapred/_HOST@EXAMPLE.COM</value>
  </property>
  <property>
    <name>mapreduce.tasktracker.keytab.file</name>
    <value>mapred.keytab</value>
  </property>
```

示例 5-12 和示例 5-13 中，配置 NodeManager 时也需要启用 `LinuxContainerExecutor`。这相当于 MR1 中的 `LinuxTaskController`。先配置 `mapred-site.xml` 文件中的参数，如示例 5-18 所示。

示例 5-18　使用 LinuxTaskController 配置 TaskTracker

```
  <property>
    <name>mapred.task.tracker.task-controller</name>
    <value>org.apache.hadoop.mapred.LinuxTaskController</value>
  </property>
  <property>
    <name>mapreduce.tasktracker.group</name>
    <value>mapred</value>
  </property>
```

还需要配置任务控制器本身。设置 `taskcontroller.cfg` 文件中的参数，确保 `mapreduce.tasktracker.group` 的值与 `mapred-site.xml` 文件中使用的值相匹配，该值通常为 `mapred`。与 `LinuxContainerExecutor` 不同，`LinuxTaskController` 不允许设置一个被允许的系统用户列表。这意味着，如果你需要允许特定的系统用户运行作业，那么可能要降低 `min.user.id` 的值，并增加 `banned.users` 列表中明确禁止的用户个数，如示例 5-19 所示。

示例 5-19　配置 LinuxTaskController

```
  mapred.local.dir=/mapred/local
  hadoop.log.dir=/var/log/hadoop-0.20-mapreduce
  mapreduce.tasktracker.group=mapred
  banned.users=root,mapred,hdfs,bin
  min.user.id=1000
```

4. Oozie

如前所述，Oozie 支持使用 Kerberos 进行身份验证。在 Oozie 中启用身份验证前，需要先配置 Oozie，使其在访问 Hadoop 时对自己进行验证。可以通过配置如下参数实现（示例 5-20 展示了一种合适的参数配置）。

`oozie.service.HadoopAccessorService.kerberos.enabled`

`hadoop.security.authentication` 设为 `kerberos` 时，该值设为 `true`。

`local.realm`

将该值设为 Hadoop 集群的默认域，这与 `krb5.conf` 文件中 `default_realm` 配置的域应该是相同的。

`oozie.service.HadoopAccessorService.kerberos.principal`

Oozie 用于进行身份验证的 Kerberos 主体，通常是 `oozie/<fqdn>@<REALM>`，其中 `<fqdn>` 是运行 Oozie 的服务器的完全限定域名（即域名全称），`<REALM>` 是本地 Kerberos 域。

`oozie.service.HadoopAccessorService.keytab.file`

keytab 文件路径，该文件含有配置的 Kerberos 主体的密钥。

示例 5-20　配置 Oozie，使之支持启用 Kerberos 的 Hadoop 集群

```
<property>
  <name>oozie.service.HadoopAccessorService.kerberos.enabled</name>
  <value>true</value>
</property>
<property>
  <name>local.realm</name>
  <value>EXAMPLE.COM</value>
</property>
<property>
  <name>oozie.service.HadoopAccessorService.keytab.file</name>
  <value>oozie.keytab</value>
</property>
<property>
  <name>oozie.service.HadoopAccessorService.kerberos.principal</name>
  <value>oozie/oozie01.example.com@EXAMPLE.COM</value>
</property>
```

Oozie 被配置为支持启用 Kerberos 的 Hadoop 集群后，即可配置 Oozie 使用 Kerberos 进行用户身份验证。相关设置如下（示例 5-21 展示了一个配置样例）。

`oozie.authentication.type`

设置用户所需的认证类型，可设为 `simple`（默认值）、`kerberos` 或实现 Hadoop `AuthenticationHandler` 接口的类的全称。

`oozie.authentication.token.validity`

认证令牌的有效时间，以秒为单位。认证令牌会在初始的认证方法（通常是 Kerberos/SPNEGO）后，以 cookie 的形式返回。

`oozie.authentication.signature.secret`

用于对认证令牌签名的密文。如果为空，则会在启动时生成一个随机密文。如果 Oozie 被配置为 HA 模式，则所有 Oozie 服务器上的该值必须相同。

`oozie.authentication.cookie.domain`

生成认证 cookie 时使用的域名，该值应当设置为集群的域名。

`oozie.authentication.kerberos.principal`

用于 Oozie 服务的 Kerberos 主体。由于 Oozie 使用 SPNEGO over HTTP 进行认证，该值必须设为 `HTTP/<fqdn>@<REALM>`，其中 `<fqdn>` 是 Oozie 服务器的域名全称，`<REALM>` 是本地 Kerberos 域。

`oozie.authentication.kerberos.keytab`

keytab 文件路径，该文件包含 Kerberos 主体的密钥。

`oozie.authentication.kerberos.name.rules`

将 Kerberos 主体转换为本地用户名的规则。该参数的格式与 Hadoop 的 `hadoop.security.auth_to_local` 参数的格式一样，如何进行配置请参考 5.1.1 节和示例 5-21。

示例 5-21　配置使用 Kerberos 认证的 Oozie

```
<property>
  <name>oozie.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>oozie.authentication.token.validity</name>
  <value>36000</value>
</property>
<property>
  <name>oozie.authentication.signature.secret</name>
  <value>FiSEcve7lBsdGpvr</value>
</property>
<property>
  <name>oozie.authentication.cookie.domain</name>
  <value>example.com</value>
</property>
<property>
  <name>oozie.authentication.kerberos.principal</name>
  <value>HTTP/oozie01.example.com@EXAMPLE.COM</value>
</property>
<property>
  <name>oozie.authentication.kerberos.principal</name>
  <value>oozie.keytab</value>
</property>
<property>
  <name>oozie.authentication.kerberos.name.rules</name>
  <value>DEFAULT</value>
</property>
```

如果以 HA 模式运行 Oozie，则需要一些额外配置。首先应当配置 `oozie-site.xml` 文件中的 `oozie.zookeeper.secure`，让 Oozie 使用 ZooKeeper ACL，如示例 5-22 所示。

示例 5-22　在 oozie-site.xml 中为 Oozie 配置 ZooKeeper ACL

```
<property>
  <name>oozie.zookeeper.secure</name>
  <value>true</value>
</property>
```

如果在 Hadoop 2.5.0 以前的版本中使用 Oozie，则需要在 HTTP 主体名称中使用负载均衡器的完全限定域名。例如，如果有运行在 `oozie01.example.com` 和 `oozie02.example.com` 上

的 Oozie 服务器，以及运行在 `oozie.example.com` 上的负载均衡器，那么应当在所有 Oozie 服务器中使用 `HTTP/oozie.example.com@EXAMPLE.COM` 作为主体。该模式下，只能通过负载均衡器访问。另外，一些诸如日志流的功能也不能用。这一步应当在 `oozie-site.xml` 文件中配置以下内容，如示例 5-23 所示。

示例 5-23　在负载均衡环境下配置 Oozie SPN

```
<property>
  <name>oozie.authentication.kerberos.principal</name>
  <value>HTTP/oozie.example.com@EXAMPLE.COM</value>
</property>
```

从 Hadoop 2.5.0 开始，可以在 Oozie 的 keytab 文件中包含多个 Kerberos 主体。这种情况下，可以在 keytab 文件中加入负载均衡器的主体和特定服务器的主体（例如 `HTTP/oozie.example.com@EXAMPLE.COM` 和 `HTTP/oozie01.example.com@EXAMPLE.COM`）。随后需要将 `oozie.authentication.kerberos.principal` 设为 `*`，如示例 5-23 所示。

示例 5-24　配置具有多个 SPN 的 Oozie

```
<property>
  <name>oozie.authentication.kerberos.principal</name>
  <value>*</value>
</property>
```

5. HBase

配置使用 Kerberos 认证的 HBase 与配置核心 Hadoop 是非常类似的。为了节约篇幅，建议参考 *The Apache HBase Reference Guide*（http://hbase.apache.org/book.html）的 Securing Apache HBase（ http://hbase.apache.org/book.html#security）章节。

5.3　小结

本章介绍了身份的概念，并展示了 Hadoop 如何使用 Kerberos 主体名映射用户名。我们还看到 Hadoop 会获取用户的组成员信息，这一点在第 6 章讲述授权时会很重要。

我们还分析了集群中进行身份验证的不同方式。虽然 Kerberos 是一种经典的例子，而且也很常用，但我们也看到，还有其他使用委托令牌进行身份验证的方式。这是 Hadoop 身份验证架构中的一个关键部分，因为它减少了完成工作流所必需的 Kerberos 认证路径的数量（例如，一个执行 Hive 查询 Oozie 工作流转化为 MapReduce 作业其实就是处理文件）。如果没有委托令牌，那么每个步骤都要请求 Kerberos 服务票据，这增加了 Kerberos KDC 的压力。

最后介绍了用户模拟的概念，讨论了系统用户如何能够代表其他用户进行身份验证。这是个常用概念，因为终端用户经常使用介于他们和他们尝试访问的服务之间的工具。使用用户模拟，系统或服务就可以使用另外一个远程服务进行身份验证，然后像终端用户直接验证一样获取访问权限。

接下来，我们将走近能够访问数据和服务的用户，接着身份验证、授权和审计的话题讨论授权。

第6章

授权

通过 5.2 节，我们了解了各种 Hadoop 生态系统项目如何支持强认证以确保用户就是他们所声称的那个人。然而，身份验证只是安全整体的一部分，你还需要对已认证用户可以执行哪些操作和访问哪些数据进行建模。这种资源保护的方式称为*授权*，这可能也是 Hadoop 安全中最复杂的话题之一。每个服务提供的服务是相对独特的，因此它们支持的授权模型也相对独特。本章将按照每个服务如何实现授权分成若干小节。

我们先看 HDFS 及其对 POSIX 文件权限的支持，以及对利用服务层授权限制用户访问特定 HDFS 功能的支持。接下来讲解 MapReduce 和 YARN，它们支持的服务层授权和用于控制系统资源访问的队列模型是类似的。对于 MapReduce 和 YARN，授权有助于安全和资源管理 / 多任务处理（关于资源管理的更多内容，推荐阅读 Eric Sammer 的《Hadoop 技术详解》）。最后介绍主流的 BigTable、Apache HBase 以及 Apache Accumulo 的授权特性，并分别讨论基于角色和基于属性的安全策略的优缺点，以及对单元级安全和列级安全进行比较。

6.1 HDFS授权

HDFS 中，每进行一次文件或目录访问都必须先经过授权检查。HDFS 采用兼容 POSIX 的文件系统中常用的授权方式。权限由三种不同类型的用户控制：**owner**（*所有者*）、**group**（*组*）和 **others**（*其他*）。每个文件或目录都归一个特定的用户所有，该用户构成了这个对象的 `owner` 类。对象还被分配了一个组，该组的所有成员构成了该对象的 `group` 类。不是对象所有者、也不属于被分配到该对象组的所有其他用户则构成 `others` 类。读、写和执行权限可被独立地授予各个类。

这些权限由一个对权限值（4 代表读，2 代表写，1 代表执行）求和后的八进制数字表示。比如要表示一个类具有一个目录的读和执行权限，则需要分配其一个八进制数字 5（4+1）。

HDFS 中，将执行权限分配给文件虽然不是非法的，但也没有意义。对于目录来说，执行位允许在知道文件名的前提下访问文件内容和元数据信息。为了列出一个目录里的所有文件名，需要该目录的读权限。

无论文件或目录的权限是什么，运行 NameNode 的用户（通常是 `hdfs`）以及 `dfs.permissions.superusergroup` 中定义的组（默认情况下是 `supergroup`）中的成员，都可以对任意文件和目录进行读、写和删除操作。就 HDFS 而言，他们相当于 Linux 系统中的 `root` 用户。

分配给 owner、group 和 others 的权限可以用依次连接的 3 个八进制值表示。例如有一个文件，owner 对其具有读和写权限，而所有其他用户对其只有读权限，那么该文件的权限就可以表示为 644。6 是分配给 owner 类的，因为其具有读和写权限（4+2）；4 是分配给 group 和 others 类的，因为他们只有读权限。对于一个所有用户都对其拥有所有权限的文件，其权限为 777。

除了标准权限之外，HDFS 还支持三种额外的特殊权限：`setuid`、`setgid` 和 `sticky`。这些权限也是用八进制数字表示的，4 代表 setuid，2 代表 setgid，1 代表 sticky。这些权限是可选的，如果使用，那么它们会包含在常规权限位的左侧。由于 HDFS 中的文件是无法执行的，因此 setuid 没有作用。setgid 同样对文件没有影响，但对于目录而言，它将新建的子文件和子目录的 group 强制设为其父节点的 group。这在 HDFS 中是默认行为，因此也没有必要刻意对目录启用 setgid。最后一个权限通常称为粘滞位，它意味着目录中的文件只能被该文件的所有者删除。没有设置粘滞位的情况下，文件可以被具有其所属目录写权限的任意用户删除。HDFS 中，无论有没有设置粘滞位，目录所有者以及 HDFS 超级管理员都可以对文件进行删除。粘滞位对于目录很有用，例如对于 `/tmp` 目录，你需要所有用户都拥有对该目录的写权限，但该目录中的数据只能被各自的所有者删除。

HDFS扩展ACL

使用 owner、group 和 world（全局）的基本 POSIX 权限，以允许访问给定的文件和目录，往往不那么简单。两个或更多不同组的用户需要访问同样的 HDFS 目录时，会怎样？如果使用基本 POSIX 权限，那么管理员有两种方案 :(1) 将该目录设为可全局访问；(2) 创建一个包含所有需要访问该目录的用户组，并分配组权限。这样做并不理想，因为方案 (1) 使得数据能够被预期之外的用户访问，而方案 (2) 从组管理的角度来说，是个很头疼的工作。当一个用户组需要读权限，而另外一个用户组需要读和写权限时，这个问题就会变得更加复杂。

随着 Hadoop 2.4 的发布，HDFS 现在配备了扩展 ACL，这些 ACL 的工作方式与 Unix 环境中扩展 ACL 的工作方式几乎一样。它允许 HDFS 中的文件和目录拥有比基本 POSIX 权限更多的权限。

要使用 HDFS 的扩展 ACL，必须先在 NameNode 上启用：将 `hdfs-site.xml` 中的 `dfs.namenode.acls.enabled` 属性设为 `true`。示例 6-1 展示了如何使用 HDFS 的扩展 ACL。

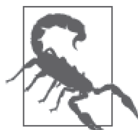
示例 6-1　HDFS 扩展 ACL

```
[alice@hadoop01 ~]$ hdfs dfs -ls /data
Found 1 items
drwxr-xr-x   - alice analysts          0 2014-10-25 19:03 /data/alice
[alice@hadoop01 ~]$ hdfs dfs -getfacl /data/alice
# file: /data/alice
# owner: alice
# group: analysts
user::rwx
group::r-x
other::r-x
[alice@hadoop01 ~]$ hdfs dfs -setfacl -m user:bob:r-x /data/alice
[alice@hadoop01 ~]$ hdfs dfs -setfacl -m group:developers:rwx /data/alice
[alice@hadoop01 ~]$ hdfs dfs -ls /data
Found 1 items
drwxr-xr-x+   - alice analysts          0 2014-10-25 19:03 /data/alice
[alice@hadoop01 ~]$ hdfs dfs -getfacl /data/alice
# file: /data/alice
# owner: alice
# group: analysts
user::rwx
user:bob:r-x
group::r-x
group:developers:rwx
mask::rwx
other::r-x
[alice@hadoop01 ~]$ hdfs dfs -chmod 750 /data/alice
[alice@hadoop01 ~]$ hdfs dfs -getfacl /data/alice
# file: /data/alice
# owner: alice
# group: analysts
user::rwx
group::r-x
group:developers:rwx    #effective:r-x
mask::r-x
other::---
[alice@hadoop01 ~]$ hdfs dfs -setfacl -b /data/alice
[alice@hadoop01 ~]$ hdfs dfs -getfacl /data/alice
# file: /data/alice
# owner: alice
# group: analysts
user::rwx
group::r-x
other::---
```

这里有几点值得强调。首先，默认情况下，文件和目录没有任何 ACL。添加 ACL 项到一个对象后，HDFS 列表会在权限列表中加入一个 `+`，例如 `drwxr-xr-x+`。另外，添加 ACL 项后，一个叫作 `mask`（掩码）的属性会列在 ACL 里，它定义了最严权限。比如用户 Bob 拥有 `rwx` 权限，但 mask 是 `r-x`，则 Bob 的有效权限是 `r-x`。`getfacl` 的输出里也会这样注明，如示例 6-1 所示。

关于 mask 的另一个重要的地方是，它会根据 ACL 中设置的最小限制权限做出调整。例如

一个 mask 目前是 r-x，为一个组加入一条新的 ACL 项以授予其 rwx 权限时，mask 就会被调整为 rwx。



在包含扩展 ACL 的文件或目录上，设置标准 POSIX 权限可能会立即对所有 ACL 项产生影响，因为 hdfs dfs -chmod 实际上会设置 mask，无论当前的 ACL 项是什么。例如，在文件或目录上设置 700 权限的结果是，除所有者之外，对所有 ACL 项的实际权限都是*禁止访问*！

例子中最后一部分展示了如何彻底移除目录的 ACL 项，只保留基本 POSIX 权限。关于扩展 ACL 的最后一点是，对于每个对象（如文件或目录），最多只能有 32 个 ACL 项。也就是说，除去 4 个项被 user、group、other 和 mask 占据，还可以添加 28 个项。超出之后，NameNode 就会抛出一个异常：setfacl: Invalid ACL: ACL has 33 entries, which exceeds maximum of 32。

另一个扩展 ACL 中有用的特性是**默认 ACL** 的使用。一条默认 ACL 只能应用到一个目录，效果是，所有在该目录中创建的子目录和子文件都会继承父目录的默认 ACL。例如，一个目录具有一条默认 ACL 项：default:group:analysts:rwx，那么该目录中创建的所有文件都会具有 group:analysts:rwx 项，而子目录会*同时*具有复制的默认 ACL 和访问 ACL。要设置默认 ACL，只需在 setfacl 命令中，于用户或组的 ACL 项前加入 default: 即可。记住，默认 ACL 自身并不实施授权，它只定义了新创建子目录和文件的继承行为。

6.2 服务级授权

Hadoop 还支持在服务级进行授权。这可以用于控制哪些用户或用户组可以访问特定协议，也可以用于阻止流氓进程伪装成守护进程。将 core-site.xml 中的 hadoop.security.authorization 项设为 true，即可启用服务级授权。实际的策略在 hadoop-policy.xml 文件中进行配置，这个文件在结构上与标准配置文件类似，每个属性用一个 property 标签定义，每个 property 标签包含一个表示属性名称和一个表示属性值的子标签。每个服务级授权属性使用一组以逗号分隔的、能够访问该协议的用户和组，以定义一个访问控制列表（ACL）。两种列表使用空格分隔，开头的空格表示空的用户列表，结尾的空格表示空的组列表。特殊值 * 表示允许所有用户访问该协议（这也是默认设置）。表 6-1 提供了一些示例 ACL。

表6-1：Hadoop访问控制列表

ACL	含义
"*"	允许所有用户
" "	不允许任何用户
"alice,bob hdusers"	允许 alice、bob 和 hdusers 组里的任意用户
"alice,bob "（末尾有空格）	允许 alice 和 bob，但不允许任何组
" hdusers"（开头有空格）“	允许 hdusers 组里的任意用户，但不允许其他用户

了解可用的 ACL 之前，我们先定义一些用户和组，以帮助指导配置。假定有一个含有少量用户和一个 Hadoop 管理员的小集群。集群用户拥有 Linux 工作站，我们想保证用户能够尽可能多地在他们的工作站上进行开发，因此不打算在工作站网络和集群之间放置防火墙。此外，假定还有一个定义了整个企业网用户和组的中心 Active Directory，集群的 KDC 被配置为使用单向信任，以允许 AD 用户不需要新凭证就能登录到集群。现在，我们想让 Hadoop 开发者能够访问集群，但又不想让整个公司都能浏览 HDFS 或者启动 MapReduce 作业。为了帮助设置，我们配置了两个组，一个为 `hadoop-users`，另一个为 `hadoop-admins`。由于这是一个新环境，我们首先只把 3 个用户——Alice、Bob 和 Joey 添到 `hadoop-users` 组。Joey 是认证过的 Hadoop 管理员，所以也将他加入 `hadoop-admins` 组。

支持服务级授权的有 HDFS、MapReduce（MR1）和 YARN（MR2）。示例中的协议 ACL 和推荐的配置值见表 6-2~表6-4，它们分别为 HDFS、MapReduce（MR1）和 YARN（MR2）而定义。其中一些属性是服务间共享的（比如刷新策略配置的协议），因此它们会在多个表中出现。由于 Hadoop 2.3 不包含 MR1，所以一些针对 MR1 策略的属性名会有所不同，部署 Hadoop 1.2 或者包含配合 HDFS 使用的 MR1 的 Hadoop 2.x 发行版时使用 MR1 属性名。

表6-2：HDFS服务级授权属性

属性名	描述	推荐值
`security.client.protocol.acl`	访问 NameNode 协议的客户端；用户代码通过 `DistributedFileSystem` 类使用	"yarn,mapred hadoop-users"
`security.client.datanode.protocol.acl`	访问 DataNode 协议的客户端	"yarn,mapred hadoop-users"
`security.get.user.mappings.protocol.acl`	获取用户映射到的组的协议	"yarn,mapred hadoop-users"
`security.datanode.protocol.acl`	访问 NameNode 协议的 DataNode	"hdfs"
`security.inter.datanode.protocol.acl`	访问 DataNode 协议的 DataNode	"hdfs"
`security.namenode.protocol.acl`	访问 NameNode 协议的 Secondary-NameNode	"hdfs"
`security.qjournal.service.protocol.acl`	访问 JournalNode 协议的 NameNode	"hdfs"
`security.zkfc.protocol.acl`	`ZKFailoverController` 暴露的协议	"hdfs"
`security.ha.service.protocol.acl`	`hdfs hadmin` 命令管理 NameNode 的 HA 状态所使用的协议	"hdfs,yarn hadoop-admins"
`security.refresh.policy.protocol.acl`	`hdfs hadmin` 命令加载最新 hadoop-policy.xml 文件所使用的协议	" hadoop-admins"
`security.refresh.user.mappings.protocol.acl`	刷新用户到组映射关系的协议	" hadoop-admins"

表6-3：MapReduce (MR1) 服务级授权属性

属性名	描述	推荐值
`security.task.umbilical.protocol.acl`	MR 任务报告任务进度所用的协议 注意：必须设为 *	"*"
`security.job.submission.protocol.acl`	客户端向 JobTracker 提交作业所用的协议	" hadoop-users"

（续）

属性名	描述	推荐值
security.inter.tracker.protocol.acl	TaskTracker 与 JobTracker 通信所用的协议	"mapred"
security.refresh.policy.protocol.acl	hadoop mradmin 命令加载最新的 hadoop-policy.xml 文件所用的协议	" hadoop-admins"
security.refresh.usertogroups.mappings.protocol.acl	刷新用户到组映射关系的协议 注意：属性名在 Hadoop 2.0 中有所改变	" hadoop-admins"
security.admin.operations.protocol.acl	hadoop mradmin 命令刷新 JobTracker 的队列和节点所用的协议	" hadoop-admins"

表 6-4：YARN和MR2的服务级授权属性

属性名	描述	推荐值
security.job.task.protocol.acl	MR 任务报告任务进度所用的协议 注意：必须设为 *	"*"
security.containermanagement.protocol.acl	ApplicationMaster 与 NodeManager 通信所用的协议 注意：必须设为 *	"*"
security.applicationmaster.protocol.acl	ApplicationMaster 与 ResourceManager 通信所用的协议 注意：必须设为 *	"*"
security.get.user.mappings.protocol.acl	获取用户映射到的组的协议	"yarn,mapred hadoop-users"
security.applicationclient.protocol.acl	客户端向 ResourceManager 提交应用所用的协议	" hadoop-users"
security.job.client.protocol.acl	作业客户端与 MR ApplicationMaster 通信所用的协议	" hadoop-users"
security.mrhs.client.protocol.acl	作业客户端与 MapReduce JobHistory Server 通信所用的协议	" hadoop-users"
security.resourcetracker.protocol.acl	访问 NodeManager 协议的 ResourceManager	"yarn"
security.resourcemanager-administration.protocol.acl	yarn rmadmin 命令管理 ResourceManager 所用的协议	"yarn"
security.resourcelocalizer.protocol.acl	ResourceLocalizationService 和 NodeManager 之间通信所用的协议	"testing"
security.ha.service.protocol.acl	yarn rmadmin 命令管理 ResourceManager 的 HA 状态所用的协议	"hdfs,yarn hadoop-admins"
security.refresh.policy.protocol.acl	yarn rmadmin 命令加载最新的 hadoop-policy.xml 文件所用的协议	" hadoop-admins"
security.refresh.user.mappings.protocol.acl	刷新用户到组映射关系的协议	" hadoop-admins"

你可能会注意到，虽然我们想让集群尽量封闭，但还是不得不配置了 4 个允许任意用户连接的协议。这么做的原因是，这些协议需要被正在运行的任务访问，这些任务会使用应用尝试（Application Attempt）或任务尝试（Task Attempt）的身份。而任务每次运行使用的身份都不一样，也与运行作业的用户名毫无关系。由于这些身份无法提前枚举，所以也无法将它们列到 ACL 或者添加到可以限制协议访问的用户组。这不是个大问题，因为这些接口被作业令牌（参见 5.2.3 节）进一步保护着，访问接口前必须提交作业令牌。

大多数协议可以分为两类：需要被客户端访问的协议；管理协议。如果想限制只有白名单里的用户或组可以使用 Hadoop，可以将客户端协议设为更严格的值。但是注意，`security.job.task.protocol.acl` (YARN/MR2) 和 `security.task.umbilical.protocol.acl` (MR1) 必须始终设为 *。这么要求是因为，使用这些协议的用户通常被设为 MapReduce 作业的作业 ID，每个作业 ID 都会变，而且也不会出现在为你的集群配置的用户组里。因此，如果这些属性设置为 * 之外的其他值，可能导致作业失败。下面看两个用户会话，首先使用 `hadoop-policy.xml` 中的默认配置（示例 6-2），然后使用表格中的推荐值（示例 6-3）。

示例 6-2　使用默认的服务级授权策略

```
[alice@hadoop01 ~]$ hdfs dfs -ls .
Found 2 items
drwx------   - alice alice          0 2014-03-29 18:59 .Trash
drwx------   - alice alice          0 2014-03-29 18:59 .staging

[alice@hadoop01 ~]$ hdfs dfs -put file.txt .

[alice@hadoop01 ~]$ hdfs dfs -rm file.txt
14/03/29 21:26:07 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop02:8020/user/alice/file.txt' to trash at:
  hdfs://hadoop02:8020/user/alice/.Trash/Current

[alice@hadoop01 ~]$ hdfs dfs -expunge
14/03/29 21:26:08 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1 minutes, Emptier interval = 0 minutes.
14/03/29 21:26:09 INFO fs.TrashPolicyDefault: Deleted trash checkpoint:
  /user/alice/.Trash/140329185911
14/03/29 21:26:09 INFO fs.TrashPolicyDefault: Created trash checkpoint:
  /user/alice/.Trash/140329212609

[alice@hadoop01 ~]$ hdfs groups
alice@CLOUDERA : alice production-etl hadoop-users

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshNodes
refreshNodes: Access denied for user alice. Superuser privilege is required

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshServiceAcl

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshUserToGroupsMappings

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshSuperUserGroupsConfiguration

[alice@hadoop01 ~]$ yarn rmadmin -refreshQueues
```

```
14/03/29 21:26:16 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -refreshNodes
14/03/29 21:26:18 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -refreshSuperUserGroupsConfiguration
14/03/29 21:26:19 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -refreshUserToGroupsMappings
14/03/29 21:26:21 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -refreshAdminAcls
14/03/29 21:26:22 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -refreshServiceAcl
14/03/29 21:26:23 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033

[alice@hadoop01 ~]$ yarn rmadmin -getGroups alice
14/03/29 21:26:25 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
alice : alice production-etl hadoop-users

[alice@hadoop01 ~]$ yarn jar /opt/cloudera/parcels/CDH/lib/
  hadoop-mapreduce/hadoop-mapreduce-examples.jar randomtextwriter random-text
14/03/29 21:26:26 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8032
Running 30 maps.
Job started: Sat Mar 29 21:26:27 EDT 2014
14/03/29 21:26:27 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8032
14/03/29 21:26:27 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN
  token 10 for alice on 172.25.2.223:8020
14/03/29 21:26:27 INFO security.TokenCache: Got dt for hdfs://hadoop02:8020;
  Kind: HDFS_DELEGATION_TOKEN, Service: 172.25.2.223:8020, Ident:
  (HDFS_DELEGATION_TOKEN token 10 for alice)
14/03/29 21:26:28 INFO mapreduce.JobSubmitter: number of splits:30
14/03/29 21:26:28 INFO mapreduce.JobSubmitter: Submitting tokens for job:
  job_1396142628007_0001
14/03/29 21:26:28 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN,
  Service: 172.25.2.223:8020, Ident: (HDFS_DELEGATION_TOKEN token 10 for alice)
14/03/29 21:26:29 INFO impl.YarnClientImpl: Submitted application
  application_1396142628007_0001
14/03/29 21:26:29 INFO mapreduce.Job: The url to track the job:
  http://hadoop02:8088/proxy/application_1396142628007_0001/
14/03/29 21:26:29 INFO mapreduce.Job: Running job: job_1396142628007_0001
14/03/29 21:26:38 INFO mapreduce.Job: Job job_1396142628007_0001 running
  in uber mode : false
14/03/29 21:26:38 INFO mapreduce.Job:  map 0% reduce 0%
14/03/29 21:28:37 INFO mapreduce.Job:  map 3% reduce 0%
```

```
14/03/29 21:28:47 INFO mapreduce.Job:  map 7% reduce 0%
14/03/29 21:28:53 INFO mapreduce.Job:  map 10% reduce 0%
14/03/29 21:29:09 INFO mapreduce.Job:  map 17% reduce 0%
14/03/29 21:29:16 INFO mapreduce.Job:  map 23% reduce 0%
14/03/29 21:29:17 INFO mapreduce.Job:  map 27% reduce 0%
14/03/29 21:29:18 INFO mapreduce.Job:  map 30% reduce 0%
14/03/29 21:29:19 INFO mapreduce.Job:  map 33% reduce 0%
14/03/29 21:29:22 INFO mapreduce.Job:  map 50% reduce 0%
14/03/29 21:29:23 INFO mapreduce.Job:  map 60% reduce 0%
14/03/29 21:29:25 INFO mapreduce.Job:  map 70% reduce 0%
14/03/29 21:29:31 INFO mapreduce.Job:  map 77% reduce 0%
14/03/29 21:30:05 INFO mapreduce.Job:  map 83% reduce 0%
14/03/29 21:30:10 INFO mapreduce.Job:  map 90% reduce 0%
14/03/29 21:30:12 INFO mapreduce.Job:  map 93% reduce 0%
14/03/29 21:30:14 INFO mapreduce.Job:  map 97% reduce 0%
14/03/29 21:30:15 INFO mapreduce.Job:  map 100% reduce 0%
14/03/29 21:30:15 INFO mapreduce.Job: Job job_1396142628007_0001
  completed successfully
14/03/29 21:30:15 INFO mapreduce.Job: Counters: 29
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=2679890
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=4550
                HDFS: Number of bytes written=33067041057
                HDFS: Number of read operations=120
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=60
        Job Counters
                Launched map tasks=30
                Other local map tasks=30
                Total time spent by all maps in occupied slots (ms)=4015333
                Total time spent by all reduces in occupied slots (ms)=0
        Map-Reduce Framework
                Map input records=30
                Map output records=49159093
                Input split bytes=4550
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=22565
                CPU time spent (ms)=808110
                Physical memory (bytes) snapshot=12234526720
                Virtual memory (bytes) snapshot=40489713664
                Total committed heap usage (bytes)=12699172864
        org.apache.hadoop.examples.RandomTextWriter$Counters
                BYTES_WRITTEN=32212265105
                RECORDS_WRITTEN=49159093
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=33067041057
Job ended: Sat Mar 29 21:30:15 EDT 2014
```

```
The job took 227 seconds.

[alice@hadoop01 ~]$ hdfs dfs -rm -r random-text
14/03/29 21:30:17 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop02:8020/user/alice/random-text' to trash at:
  hdfs://hadoop02:8020/user/alice/.Trash/Current

[alice@hadoop01 ~]$ hdfs dfs -expunge
14/03/29 21:30:18 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1 minutes, Emptier interval = 0 minutes.
14/03/29 21:30:19 INFO fs.TrashPolicyDefault: Deleted trash checkpoint:
  /user/alice/.Trash/140329212609
14/03/29 21:30:19 INFO fs.TrashPolicyDefault: Created trash checkpoint:
  /user/alice/.Trash/140329213019
```

示例 6-2 展示了 Alice 在使用一系列用户和管理命令。使用默认的服务级授权策略时，虽然有一些命令（如 `hdfs dfsadmin -refreshNodes`）需要超级用户权限，但很多命令使用默认的服务级授权策略时，并不需要特殊权限。示例 6-3 使用前面介绍的推荐策略执行了完全相同的一系列命令。

示例 6-3　使用推荐的服务级授权策略

```
[alice@hadoop01 ~]$ hdfs dfs -ls .
Found 2 items
drwx------   - alice alice          0 2014-03-29 18:52 .Trash
drwx------   - alice alice          0 2014-03-29 18:45 .staging

[alice@hadoop01 ~]$ hdfs dfs -put file.txt .

[alice@hadoop01 ~]$ hdfs dfs -rm file.txt
14/03/29 18:54:11 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop02:8020/user/alice/file.txt' to trash at:
  hdfs://hadoop02:8020/user/alice/.Trash/Current

[alice@hadoop01 ~]$ hdfs dfs -expunge
14/03/29 18:54:13 INFO fs.TrashPolicyDefault: Namenode trash configuration:
  Deletion interval = 1 minutes, Emptier interval = 0 minutes.
14/03/29 18:54:13 INFO fs.TrashPolicyDefault: Deleted trash checkpoint:
  /user/alice/.Trash/140329185237
14/03/29 18:54:13 INFO fs.TrashPolicyDefault: Created trash checkpoint:
  /user/alice/.Trash/140329185413

[alice@hadoop01 ~]$ hdfs groups
alice@CLOUDERA : alice production-etl hadoop-users

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshNodes
refreshNodes: Access denied for user alice. Superuser privilege is required

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshServiceAcl
refreshServiceAcl: User alice@CLOUDERA (auth:KERBEROS) is not authorized for
  protocol interface
  org.apache.hadoop.security.authorize.RefreshAuthorizationPolicyProtocol,
  expected client Kerberos principal is null
```

```
[alice@hadoop01 ~]$ hdfs dfsadmin -refreshUserToGroupsMappings
refreshUserToGroupsMappings: User alice@CLOUDERA (auth:KERBEROS) is not
  authorized for protocol interface
  org.apache.hadoop.security.RefreshUserMappingsProtocol, expected client
  Kerberos principal is null

[alice@hadoop01 ~]$ hdfs dfsadmin -refreshSuperUserGroupsConfiguration
refreshSuperUserGroupsConfiguration: User alice@CLOUDERA (auth:KERBEROS) is
  not authorized for protocol interface
  org.apache.hadoop.security.RefreshUserMappingsProtocol, expected client
  Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshQueues
14/03/29 18:54:21 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
refreshQueues: User alice@CLOUDERA (auth:KERBEROS) is not authorized
  for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshNodes
14/03/29 18:54:22 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
refreshNodes: User alice@CLOUDERA (auth:KERBEROS) is not authorized
  for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshSuperUserGroupsConfiguration
14/03/29 18:54:24 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
refreshSuperUserGroupsConfiguration: User alice@CLOUDERA (auth:KERBEROS)
  is not authorized for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshUserToGroupsMappings
14/03/29 18:54:25 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
refreshUserToGroupsMappings: User alice@CLOUDERA (auth:KERBEROS)
  is not authorized for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshAdminAcls
14/03/29 18:54:26 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
refreshAdminAcls: User alice@CLOUDERA (auth:KERBEROS)
  is not authorized for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -refreshServiceAcl
14/03/29 18:54:28 INFO client.RMProxy: Connecting to ResourceManager at
```

```
  hadoop02/172.25.2.223:8033
refreshServiceAcl: User alice@CLOUDERA (auth:KERBEROS)
  is not authorized for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn rmadmin -getGroups alice
14/03/29 18:54:29 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8033
getGroups: User alice@CLOUDERA (auth:KERBEROS)
  is not authorized for protocol interface
  org.apache.hadoop.yarn.server.api.ResourceManagerAdministrationProtocolPB,
  expected client Kerberos principal is null

[alice@hadoop01 ~]$ yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/
  hadoop-mapreduce-examples.jar randomtextwriter random-text
14/03/29 18:54:31 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8032
Running 30 maps.
Job started: Sat Mar 29 18:54:32 EDT 2014
14/03/29 18:54:32 INFO client.RMProxy: Connecting to ResourceManager at
  hadoop02/172.25.2.223:8032
14/03/29 18:54:32 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN
  token 9 for alice on 172.25.2.223:8020
14/03/29 18:54:32 INFO security.TokenCache: Got dt for hdfs://hadoop02:8020;
  Kind: HDFS_DELEGATION_TOKEN, Service: 172.25.2.223:8020, Ident:
  (HDFS_DELEGATION_TOKEN token 9 for alice)
14/03/29 18:54:32 INFO mapreduce.JobSubmitter: number of splits:30
14/03/29 18:54:32 INFO mapreduce.JobSubmitter: Submitting tokens for job:
  job_1396131817617_0003
14/03/29 18:54:32 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN,
  Service: 172.25.2.223:8020, Ident: (HDFS_DELEGATION_TOKEN token 9 for alice)
14/03/29 18:54:33 INFO impl.YarnClientImpl: Submitted application
  application_1396131817617_0003
14/03/29 18:54:33 INFO mapreduce.Job: The url to track the job:
  http://hadoop02:8088/proxy/application_1396131817617_0003/
14/03/29 18:54:33 INFO mapreduce.Job: Running job: job_1396131817617_0003
14/03/29 18:54:40 INFO mapreduce.Job: Job job_1396131817617_0003
  running in uber mode : false
14/03/29 18:54:40 INFO mapreduce.Job:  map 0% reduce 0%
14/03/29 18:56:20 INFO mapreduce.Job:  map 3% reduce 0%
14/03/29 18:56:53 INFO mapreduce.Job:  map 7% reduce 0%
14/03/29 18:56:57 INFO mapreduce.Job:  map 10% reduce 0%
14/03/29 18:56:59 INFO mapreduce.Job:  map 13% reduce 0%
14/03/29 18:57:02 INFO mapreduce.Job:  map 17% reduce 0%
14/03/29 18:57:15 INFO mapreduce.Job:  map 20% reduce 0%
14/03/29 18:57:36 INFO mapreduce.Job:  map 27% reduce 0%
14/03/29 18:57:44 INFO mapreduce.Job:  map 30% reduce 0%
14/03/29 18:57:59 INFO mapreduce.Job:  map 33% reduce 0%
14/03/29 18:58:09 INFO mapreduce.Job:  map 37% reduce 0%
14/03/29 18:58:19 INFO mapreduce.Job:  map 40% reduce 0%
14/03/29 18:58:23 INFO mapreduce.Job:  map 43% reduce 0%
14/03/29 18:58:25 INFO mapreduce.Job:  map 47% reduce 0%
14/03/29 18:58:35 INFO mapreduce.Job:  map 50% reduce 0%
14/03/29 18:58:36 INFO mapreduce.Job:  map 53% reduce 0%
```

```
14/03/29 18:58:39 INFO mapreduce.Job:  map 57% reduce 0%
14/03/29 18:58:40 INFO mapreduce.Job:  map 60% reduce 0%
14/03/29 18:58:44 INFO mapreduce.Job:  map 63% reduce 0%
14/03/29 18:58:45 INFO mapreduce.Job:  map 67% reduce 0%
14/03/29 18:58:47 INFO mapreduce.Job:  map 70% reduce 0%
14/03/29 18:58:53 INFO mapreduce.Job:  map 73% reduce 0%
14/03/29 18:58:55 INFO mapreduce.Job:  map 80% reduce 0%
14/03/29 18:58:57 INFO mapreduce.Job:  map 83% reduce 0%
14/03/29 18:59:01 INFO mapreduce.Job:  map 90% reduce 0%
14/03/29 18:59:05 INFO mapreduce.Job:  map 93% reduce 0%
14/03/29 18:59:07 INFO mapreduce.Job:  map 100% reduce 0%
14/03/29 18:59:07 INFO mapreduce.Job: Job job_1396131817617_0003
  completed successfully
14/03/29 18:59:07 INFO mapreduce.Job: Counters: 29
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=2679890
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=4550
                HDFS: Number of bytes written=33067034387
                HDFS: Number of read operations=120
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=60
        Job Counters
                Launched map tasks=30
                Other local map tasks=30
                Total time spent by all maps in occupied slots (ms)=5319195
                Total time spent by all reduces in occupied slots (ms)=0
        Map-Reduce Framework
                Map input records=30
                Map output records=49157281
                Input split bytes=4550
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=13711
                CPU time spent (ms)=741910
                Physical memory (bytes) snapshot=10065694720
                Virtual memory (bytes) snapshot=40491339776
                Total committed heap usage (bytes)=14946533376
        org.apache.hadoop.examples.RandomTextWriter$Counters
                BYTES_WRITTEN=32212267432
                RECORDS_WRITTEN=49157281
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=33067034387
Job ended: Sat Mar 29 18:59:07 EDT 2014
The job took 275 seconds.

[alice@hadoop01 ~]$ hdfs dfs -rm -r random-text
14/03/29 18:59:09 INFO fs.TrashPolicyDefault: Namenode trash
  configuration: Deletion interval = 1440 minutes, Emptier
```

```
  interval = 0 minutes.
Moved: 'hdfs://hadoop02:8020/user/alice/random-text' to trash
  at: hdfs://hadoop02:8020/user/alice/.Trash/Current

[alice@hadoop01 ~]$ hdfs dfs -expunge
14/03/29 18:59:10 INFO fs.TrashPolicyDefault: Namenode trash
  configuration: Deletion interval = 1 minutes, Emptier
  interval = 0 minutes.
14/03/29 18:59:11 INFO fs.TrashPolicyDefault: Deleted trash checkpoint:
  /user/alice/.Trash/140329185413
14/03/29 18:59:11 INFO fs.TrashPolicyDefault: Created trash checkpoint:
  /user/alice/.Trash/140329185911
```

这次 Alice 仍然能够访问所有用户功能，但管理员功能被禁止了，报错信息为 `User alice@CLOUDERA (auth:KERBEROS) is not authorized for protocol interface` < *协议名* >。这表示该用户既不在该协议的 ACL 表里，也不属于 ACL 表中的某个组。服务级授权虽然复杂，却是一种控制 Hadoop 集群访问的有力工具。例如，在我们所配的策略控制下，`hdfs` 用户不再具有查看或修改 HDFS 中文件的权限，除非它被添加到 `hadoop-users` 组。这对于需要由某个管理动作追溯执行该动作的管理员的公司来说，非常有用。如果将 `dfs.permissions.superusergroup` 设置为 `hadoop-admins`，以结合服务级授权，就能将管理动作绑定到特定账号。示例 6-4 展示了 `hdfs` 用户尝试列出 Alice 主目录下的文件以及删除她所上传的一个文件时，将会如何。

示例 6-4　用户 hdfs 被拒绝访问 ClientProtocol

```
[hdfs@hadoop01 ~]$ hdfs dfs -ls /user/alice/
ls: User hdfs@CLOUDERA (auth:KERBEROS) is not authorized for protocol interface
  org.apache.hadoop.hdfs.protocol.ClientProtocol, expected client Kerberos principal
is null
[hdfs@hadoop01 ~]$ hdfs dfs -rm /user/alice/file.txt
rm: User hdfs@CLOUDERA (auth:KERBEROS) is not authorized for protocol interface
  org.apache.hadoop.hdfs.protocol.ClientProtocol, expected client Kerberos principal
is null
```

注意，用户 `hdfs` 在协议层就被拒绝访问了，这发生在 HDFS 层执行的权限检查之前。虽然从文件系统的角度看，`hdfs` 是超级用户，但根据此前的服务级检查，它不能查看和修改任何数据。示例 6-5 展示了一个 `hadoop-admins` 组成员 Joey 尝试执行相同的动作时，情况会如何。

示例 6-5　hadoop-admins 组的一个成员删除用户文件

```
[joey@hadoop01 ~]$ hdfs dfs -ls /user/alice/
Found 3 items
drwx------    - alice alice         0 2014-03-29 21:30 /user/alice/.Trash
drwx------    - alice alice         0 2014-03-29 21:30 /user/alice/.staging
-rw-------     3 alice alice        5 2014-03-29 21:48 /user/alice/file.txt
[joey@hadoop01 ~]$ hdfs dfs -rm /user/alice/file.txt
14/03/29 21:49:26 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion
  interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop02:8020/user/alice/file.txt' to trash at:
```

```
  hdfs://hadoop02:8020/user/joey/.Trash/Current
[joey@hadoop01 ~]$ hdfs groups joey
joey : joey hadoop-admins hadoop-users
[joey@hadoop01 ~]$ hdfs groups hdfs
hdfs : hdfs hadoop
```

你会注意到，这次这些行为被允许了。因为 Joey 同时是 `hadoop-admins` 组（被配置为 HDFS 中的超级用户组）和 `hadoop-users` 组（使用户能够访问 HDFS 客户端协议）的成员。

除了在 `hadoop-policy.xml` 配置 ACL 之外，某些 HDFS 管理动作（如强制执行 HA 故障转移）也只对 HDFS 集群管理员可用。管理员可以使用 `hdfs-site.xml` 中的 `dfs.cluster.administrators` 进行配置，其值为以逗号分隔的、可以管理 HDFS 的用户列表和组列表，两个列表之间用空格分隔。开头的空格表示空的用户列表，结尾的空格表示空的组列表。特殊值 * 表示所有用户具有对 HDFS 的管理权限，值 " "（不包含引号）表示所有用户都没有管理权限（这是默认设置）。示例 6-6 给出了我们的示例环境下推荐的配置。

示例 6-6　hdfs-site.xml 中的 dfs.cluster.administrators 配置

```
<property>
  <name>dfs.cluster.administrators</name>
  <value>hdfs hadoop-admins</value>
</property>
```

管理 MapReduce Job History Server 的 ACL 不在 `hadoop-policy.xml` 文件里配置，而是要配置 `mapred-site.xml` 中的 `mapreduce.jobhistory.admin.acl`，其值为逗号分隔的用户列表和组列表，格式与本节开始部分以及表 6-1 中描述的一致。特殊值 * 表示允许所有用户管理 JobHistory Server（这是默认配置）。示例 6-7 给出了推荐的配置。

示例 6-7　mapred-site.xml 中的 mapreduce.jobhistory.admin.acl 配置

```
<property>
  <name>mapreduce.jobhistory.admin.acl</name>
  <value>mapred hadoop-admins</value>
</property>
```

6.3　MapReduce和YARN的授权

MapReduce 和 YARN 都不控制对数据的访问，但都提供了对集群资源（如 CPU、内存、硬盘 I/O 和网络 I/O）的访问。由于这些资源是有限的，因此管理员通常要为特定用户或组分配资源，尤其在多用户环境中更是如此。上一节描述的服务级授权控制了对特定协议的访问，比如谁可以向集群提交作业而谁不可以，但对于控制集群资源的访问还不够细化。MapReduce（MR1）和 YARN 都支持以作业队列的方式限制如何给作业分配资源。为了安全地控制这些资源，Hadoop 支持作业队列上的访问控制列表（ACL）。这些 ACL 控制了哪些用户可以提交到特定队列，以及哪些用户可以管理队列。MapReduce 定义了不同类型的用户，这影响 ACL 的解释方式。

MapReduce/YARN 集群所有者

启动 JobTracker 进程（MR1）或 ResourceManager 进程（YARN）的用户被定义为集群所有者。该用户拥有提交作业到任意队列的权限，并能管理任意队列或作业。大多数情况下，集群所有者在 MapReduce（MR1）中即是 `mapred`，在 YARN 中即是 `yarn`。由于以集群所有者的身份运行作业很危险，因此 LinuxTaskController 默认把 `mapred` 和 `yarn` 用户列入黑名单，使其无法提交作业。

MapReduce 管理员

可以通过设置以创建与集群所有者权限相同的全局 MapReduce 管理员。定义特定的用户或组为管理员的好处是，可以对每个管理员各自的动作进行审计。这也能避免不得不将密码分配到共享账号而导致被破解机率增大的情况。

作业所有者

作业所有者是提交该作业的用户。作业所有者总是可以管理他们自己的作业，但只有被授予作业提交权限的用户才能提交作业到队列。

队列管理员

用户或组可以被授予对队列中所有作业的管理权限，队列管理员也可以提交作业到他所管理的队列。

6.3.1 MapReduce（MR1）

对于 MR1，ACL 是全局管理的，并应用于任意支持 ACL 的作业调度器。CapacityScheduler 和 FairScheduler 都支持 ACL，但 FIFO（默认的）调度器不支持。配置单队列 ACL 前，需要启用 MapReduce ACL，配置 MapReduce 管理员，以及在 `mapred-site.xml` 中定义队列名。

`mapred.acls.enabled`

设为 `true` 时，提交或管理作业后会检查 ACL。在 JobTracker 接口中对查看和修改作业进行授权时，也要检查 ACL。

`mapreduce.cluster.administrators`

为 MapReduce 集群配置管理员。无论作业或队列相关的 ACL 如何配置，集群管理员总可以管理任意作业或队列。该配置的格式是一个逗号分隔的、可以访问该协议的用户列表和组列表。两个列表之间使用空格分隔，开头的空格表示空的用户列表，结尾的空格表示空的组列表。特殊值 `*` 表示允许所有用户访问该协议（这是默认配置）。具体示例见表 6-1。

`mapred.queue.names`

一系列以逗号分隔的队列名。如果要为某个队列配置 ACL，则该队列必须被列在该属性里。MapReduce 通常支持至少一个名为 `default` 的队列，因此该参数应当始终在定义的队列列表中包含 `default`。

单队列 ACL 配置存储在 `mapred-queue-acls.xml` 里。每个队列有两种 ACL 可以配置，分别是提交 ACL 和管理 ACL。

`mapred.queue.<queue_name>.acl-submit-job`

可以向队列 `queue_name` 提交作业的用户的访问控制表。提交作业的 ACL 的格式为逗号分隔的、允许向该队列提交作业的用户列表和组列表，其格式与 6.2 节和表 6-1 中描述的一致。特殊值 `*` 表示允许所有用户访问该协议（这是默认设置）。但无论该项如何设置，集群所有者和 MapReduce 管理员都可以提交作业。

`mapred.queue.<queue_name>.acl-administer-jobs`

可以对队列 `queue_name` 中的所有作业进行查看详情、终止或修改优先级操作的用户的访问控制表。管理作业的 ACL 的格式为逗号分隔的、允许管理该队列中作业的用户列表和组列表，其格式与 6.2 节和表 6-1 中描述的一致。特殊值 `*` 表示允许所有用户访问该协议（这是默认设置）。但无论该项如何设置，集群所有者和 MapReduce 管理员都可以管理所有队列中的所有作业。作业所有者也可以管理作业。

除单队列 ACL 外，还有两种可以为每个作业配置的 ACL。这些设置的默认值可以放在客户端使用的 `mapred- site.xml` 文件里，这些值也可以被某个作业重写。

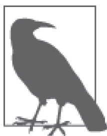
`mapreduce.job.acl-view-job`

允许查看作业详情的用户的访问控制表。查看作业的 ACL 的格式为逗号分隔的、允许查看作业详情的用户列表和组列表，其格式与 6.2 节和表 6-1 中描述的一致。特殊值 `*` 表示允许所有用户访问该协议（这是默认设置）。但无论该项如何设置，集群所有者、MapReduce 管理员和作业提交到的队列的管理员都可以查看作业。该 ACL 控制了对作业级计数器、任务级计数器、任务的诊断信息、TaskTracker Web 界面展示的任务日志以及 JobTracker Web 界面展示的 `job.xml` 的访问。

`mapreduce.job.acl-modify-job`

允许停止作业、结束任务、放弃任务、设置作业优先级的用户的访问控制表。修改作业的 ACL 的格式为逗号分隔的、允许修改作业的用户列表和组列表，其格式与 6.2 节和表 6-1 中描述的一致。特殊值 `*` 表示允许所有用户访问该协议（这是默认设置）。但无论该项如何设置，作业所有者、集群所有者、MapReduce 管理员和作业提交到的队列的管理员都可以修改作业。

如果有这么一个部署场景，你想为访问作业详情配置一个默认拒绝的策略，那么对于这两种配置而言，一个明智的默认值就是一个空格 `" "`（不包含引号）。这会禁止除作业所有者、队列管理员、集群管理员和集群所有者之外的所有用户访问作业细节。



为了控制对 JobTracker Web 界面中作业详情的访问，必须如前所述配置 MapReduce ACL，也需要按照第 11 章描述的方法启用 Web 界面的授权。

6.3.2 YARN (MR2)

YARN/MR2 中，队列 ACL 不再是全局定义的，每个调度器都提供了自己定义 ACL 的方法。ACL 仍然是全局启用的，有一个全局的 ACL 定义 YARN 管理员。启用 YARN ACL 以及定义管理员的选项在 `yarn-site.xml` 里进行配置。示例 6-8 提供了一些配置示例。

示例 6-8　yarn-site.xml 中的 YARN ACL 配置

```
<property>
  <name>yarn.acl.enable</name>
  <value>true</value>
</property>
<property>
  <name>yarn.admin.acl</name>
  <value>yarn hadoop-admins</value>
</property>
```

由于每个调度器的配置各不相同，我们将逐个梳理如何设置队列 ACL。对于这两种例子，我们会实现相同的用例。我们的集群主要用于运行生产 ETL 流水线和用于生成日常报告的生产查询，也有一些临时报告，但生产作业应优先处理。为了进行访问控制，我们额外定义了两个用户组，这两个组只包含先前定义的 `hadoop-users` 的一部分。一个是 `production-etl` 组，它包含了运行生产 ETL 作业的用户；另一个是 `production-queries` 组，它包含了运行生产查询的用户。该例子中，Alice 是 `production-etl` 组的成员，Bob 是 `production-queries` 组的成员。先配置 FairScheduler。

1. FairScheduler

为了保证生产作业所需的资源，首先必须禁用 FairScheduler 的默认行为，即将每个用户放到与其用户名相匹配的队列。可以通过设置两个参数 `yarn.scheduler.fair.user-as-default_queue` 和 `yarn.scheduler.fair.allow-undeclared-pools` 的值为 `false` 实现。第一个参数将默认的队列改为 `default`，第二个参数确保用户无法向未预定义的队列提交作业。这些设置以及开启 FairScheduler 的设置项在示例 6-9 中都能找到。

示例 6-9　yarn-site.xml 中的 FairScheduler 配置

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>
    org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler
  </value>
</property>
<property>
  <name>yarn.scheduler.fair.user-as-default-queue</name>
  <value>false</value>
</property>
<property>
  <name>yarn.scheduler.fair.allow-undeclared-pools</name>
  <value>false</value>
</property>
```

接下来，在 `fair-scheduler.xml` 文件中定义队列及其 ACL。FairScheduler 使用分级队列系统，每个队列都是 `root` 队列的子队列。示例中，我们想将集群资源中的 90% 分配给生产作业，10% 分配给临时作业。为了实现这点，我们定义了两个 root 队列的子队列：生产作业的是 `prod`，临时作业的是 `default`。之所以将临时队列称为 default，是因为如果没有为某个作业指定队列，那么该作业就会被提交到这里。资源管理是个复杂的话题，我们可以调节各种各样的设置，以井然有序地控制资源。由于我们关注的是安全方面，因此将用一个简化的方式使用队列权重控制资源。需要了解的是，所有队列具有相同的父队列，它们

的资源配置被定义为其权重除以其所有兄弟队列的权重。该例子中，我们可以为 prod 分配权重 9.0，为 default 分配权重 1.0，这样就能得到 90 ： 10 的划分比例。

我们还想将生产队列分成两个子队列：一个用于 ETL 作业，一个用于请求。本例中，我们将把两个队列的权重都设为 1.0，因此两个队列的权重是一样的。值得注意的是，份额的计算发生在父队列上下文中。本例中，意思就是由于我们将 etl 和 queries 的队列都设成 prod 队列 50% 的资源，最终它们获得的资源是全局等额分量，每个队列是 45%（50% × 90% = 45%）。

资源是继承的，ACL 也是。使用 FairScheduler 后，能提交作业到某个队列的用户也能提交作业到其子队列，这也同样适用于队列的管理权限。和前面的例子一样，我们想使 hadoop-admins 组的任意成员都能管理任意作业 / 队列，就要将他们加到根队列的 aclAdministerApps ACL 中。值得注意的是，必须将 aclSubmitApps ACL 设成 " "（不含引号），否则，由于没有定义的默认 ACL 允许所有，因此会导致所有用户都能提交作业到任意队列。对于默认队列，我们想要允许 hadoop-users 组的任意成员都能提交作业，因此要将 aclSubmitApps 设成 " hadoop-users"（不含引号，并注意开头有空格）。对于 prod.etl 和 prod.queries 队列，则要将 aclSubmitApps 分别设为 " production-etl" 和 " production-queries"（不含引号），这些都是我们之前定义的组。完整的 FairScheduler 配置如示例 6-10 所示。

示例 6-10　fair-scheduler.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<allocations>
    <queue name="root">
        <weight>1.0</weight>
        <aclSubmitApps> </aclSubmitApps>
        <aclAdministerApps> hadoop-admins</aclAdministerApps>
        <queue name="prod">
            <weight>9.0</weight>
            <aclSubmitApps> </aclSubmitApps>
            <aclAdministerApps> </aclAdministerApps>
            <queue name="etl">
                <weight>1.0</weight>
                <aclSubmitApps> production-etl</aclSubmitApps>
                <aclAdministerApps>alice </aclAdministerApps>
            </queue>
            <queue name="queries">
                <weight>1.0</weight>
                <aclSubmitApps> production-queries</aclSubmitApps>
                <aclAdministerApps>bob </aclAdministerApps>
            </queue>
        </queue>
        <queue name="default">
            <weight>1.0</weight>
            <aclSubmitApps> hadoop-users</aclSubmitApps>
            <aclAdministerApps> </aclAdministerApps>
        </queue>
    </queue>
</allocations>
```

现在看看 Bob 在没有预先定义队列 ACL 的情况下，尝试终止一个 Alice 的作业会发生什么。首先，Bob 得到运行着的作业列表，并找到 Alice 作业的 `JobId`，然后发送终止作业的请求。由于缺乏对谁能管理作业而谁不能管理作业的控制，YARN 会很乐意满足他的请求。示例 6-11 给出了 Bob 的完整用户会话。

示例 6-11　未定义 ACL 的情况下终止其他用户的作业

```
[bob@hadoop01 ~]$ mapred job -list
Total jobs:1
                  JobId      State           StartTime      UserName
 job_1396200012809_0002    RUNNING       1396201153018         alice
[bob@hadoop01 ~]$ mapred job -kill job_1396200012809_0002
Killed job job_1396200012809_0002
```

这并不理想，因为用户可以干扰其他用户的生产作业。更重要的是，一个简单的复制 / 粘贴错误就有可能导致一个用户不小心终止另外一个用户的作业。如果我们在 FairScheduler 中配置了 ACL 之后再执行相同的操作，就会得到示例 6-12 所示的结果。

示例 6-12　Bob 被队列 ACL 拒绝给予管理权限

```
[bob@hadoop01 ~]$ mapred job -list
Total jobs:1
                  JobId      State           StartTime        UserName
 job_1396192703139_0001    RUNNING       1396192707596           alice
[bob@hadoop01 ~]$ mapred job -kill job_1396192703139_0001
...
Exception in thread "main" java.io.IOException: org.apache.hadoop.yarn.exceptions.Yar
nException: java.security.AccessControlException: User bob cannot perform operation M
ODIFY_APP on application_1396192703139_0001
        at org.apache.hadoop.yarn.ipc.RPCUtil.getRemoteException(RPCUtil.java:38)
...
[bob@hadoop01 ~]$
```

很多时候，一些管理员必须要能终止另外一个用户的作业，因此我们在根队列上将管理权限设置到 `hadoop-admins` 组。因此，如果其中一个管理员 Joey 尝试终止一个作业，系统将进行以下处理，如示例 6-13 所示。

示例 6-13　成功终止一个 MapReduce 作业

```
[joey@hadoop01 ~]$ mapred job -list
Total jobs:1
                  JobId      State           StartTime        UserName
 job_1396192703139_0002    RUNNING       1396193202565           alice
[joey@hadoop01 ~]$ mapred job -kill job_1396192703139_0002
Killed job job_1396192703139_0002
```

控制管理权限显然是有用的，但防止用户提交作业到错误队列也很重要。我们的例子中，由于 Alice 是 `production-etl` 组的成员，因此她具有提交作业到 `prod.etl` 队列的权限。但她不是 `production-queries` 组的成员，因此如果尝试提交作业到该处，则会被拒绝，如示例 6-14 所示。

示例 6-14　Alice 被拒绝向 prod.queries 队列提交作业

```
[alice@hadoop01 ~]$ yarn jar \
  /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar \
  randomtextwriter -Dmapreduce.job.queuename=prod.queries random-text
...
Job started: Sun Mar 30 13:20:57 EDT 2014
14/03/30 13:20:59 ERROR security.UserGroupInformation: PriviledgedActionException
as:alice@CLOUDERA (auth:KERBEROS) cause:java.io.IOException: Failed to run job :
User alice cannot submit applications to queue root.prod.queries
...
```

2. CapacityScheduler

CapacityScheduler 与 FairScheduler 一样支持分级队列，也支持与 FairScheduler 一样的单队列 ACL 和 ACL 继承策略。实际上，从安全角度看，这两个调度器是相同的，唯一的区别在于配置文件的格式。为了实现与前所述相同的策略，需要先在 `yarn-site.xml` 中启用 CapacityScheduler，如示例 6-15 所示。

示例 6-15　yarn-site.xml 中的 CapacityScheduler 配置

```
<property>
 <name>
  yarn.resourcemanager.scheduler.class
 </name>
 <value>
  org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler
</value>
</property>
```

一旦启用，CapacityScheduler 就会从 `capacity-scheduler.xml` 文件读取配置。示例 6-16 展示了实现同一队列和 ACL 的配置。对于 FairScheduler，ACL 被配置为队列定义的一个子元素，用 `aclSubmitApps` 标签控制谁可以提交应用到队列，用 `aclAdministerApps` 标签控制谁可以管理队列中的作业。对于 CapacityScheduler 来说，对应的设置分别是如下所示的属性（将 `<path-to-queue >` 替换为队列的层次结构）。

- `yarn.scheduler.capacity.root.<` *`path-to-queue`* `>.acl_submit_applications`
- `yarn.scheduler.capacity.root.<` *`path-to-queue`* `>.acl_administer_applications`

例如，定义 prod.etl 队列的 ACL 是 `yarn.scheduler.capacity.root.prod.etl.acl_submit_applications`，如示例 6-16 所示。

示例 6-16　capacity-scheduler.xml

```
<!-- Define ACLs and subqueues for the root queue -->
<property>
  <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
  <value> </value>
</property>
<property>
  <name>yarn.scheduler.capacity.root.acl_administer_applications</name>
```

```
    <value> hadoop-admins</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>prod,default</value>
  </property>

  <!-- Define capacity and ACLs for the root.default queue -->
  <property>
    <name>yarn.scheduler.capacity.root.default.capacity</name>
    <value>10.0</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
    <value> hadoop-users</value>
  </property>

  <!-- Define capacity, ACLs, and subqueues for the root.prod queue -->
  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>90.0</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.queues</name>
    <value>etl,queries</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.acl_submit_applications</name>
    <value> </value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.acl_administer_applications</name>
    <value> </value>
  </property>

  <!-- Define capacity and ACLs for the root.prod.etl queue -->
  <property>
    <name>yarn.scheduler.capacity.root.prod.etl.capacity</name>
    <value>50.0</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.etl.acl_submit_applications</name>
    <value> production-etl</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.etl.acl_administer_applications</name>
    <value>alice </value>
  </property>

  <!-- Define capacity and ACLs for the root.prod.queries queue -->
  <property>
    <name>yarn.scheduler.capacity.root.prod.queries.capacity</name>
    <value>50.0</value>
  </property>
  <property>
```

```
  <name>yarn.scheduler.capacity.root.prod.queries.acl_submit_applications</name>
  <value> production-queries</value>
</property>
<property>
  <name>yarn.scheduler.capacity.root.prod.queries.acl_administer_applications</name>
  <value>bob </value>
</property>
```

6.4 ZooKeeper ACLs

Apache ZooKeeper 使用 ACL 控制对 ZNode（路径）的访问。ZooKeeper 的 ACL 与 POSIX 权限位类似，但更加灵活，因为其权限设置在单用户上，而不是所有者和主要组上。实际上，ZooKeeper 里没有所有者或组的概念。5.2.2 节讲过，用户由一个身份验证模式和模式对应的 ID 确定，ID 的格式根据模式的不同而不同。

每个 ACL 包含一个模式、ID 和权限。可用的权限列表见表 6-5。值得注意的是，ZooKeeper 中，权限是非递归的，只作用于所属的 ZNode，不作用于其子节点。由于 ZooKeeper 中没有 ZNode 所有者的概念，因此用户需要有 ZNode 的 `ADMIN` 权限才能设置 ACL。

表6-5：ZooKeeper ACL权限

权限	描述
CREATE	创建子 Znode 的权限
READ	读取 ZNode 数据以及列出 ZNode 子节点的权限
WRITE	设置 ZNode 数据的权限
DELETE	删除子 Znode 的权限
ADMIN	设置 ACL 的权限

`CREATE` 和 `DELETE` 权限用于控制谁可以创建 ZNode 的子节点。授予 `CREATE` 权限但不授予 `DELETE` 权限的使用情况是：你希望用户可以在某个路径创建子节点，但该路径下只有管理员才能删除子节点。

如果使用 JAVA API 添加 ACL，首先要使用模式项和 ID 号创建一个 `ID` 对象，然后使用该 `ID` 以及整型的权限创建一个 `ACL` 对象。你可以手动计算权限的整型值，或者使用 `ZooDefs.Perms` 类中的常量得出想要设置的权限组合而成的整型值。如示例 6-17 所示，使用 Java 代码在某个路径上设置 ACL。

示例 6-17　使用 JAVA API 设置 ZooKeeper ACL

```
// 连接到ZooKeeper
ZooKeeper zk = new ZooKeeper("zk.example.com:2181", 60000, watcher);

// 使用密码secret为alice创建Id
Id id = new Id("digest", "alice:secret");

// 创建授予Alice READ和CREATE权限的ACL
ACL acl = new ACL(ZooDefs.Perms.READ | ZooDefs.Perms.CREATE, id);

zk.setAcl("/test", Arrays.asList(new ACL[] {acl}), -1);
```

5.2.2 节已经描述过 `digest` 模式，但 ZooKeeper 还支持一系列其他内置模式。表 6-6 描述了可用模式，以及 ACL 中使用的 ID 的格式。对于 `world` 模式，唯一的 ID 就是字符串 `anyone`。`digest` 模式使用 <*用户名*>:<*密码*> 的 sha1 值的 base64 编码。`ip` 模式允许通过某个 IP 地址或 CIDR 标记的某段 IP 地址设置 ACL。最后，`sasl` 模式使用 <*主体*> 作为 ID，默认情况下，该主体即是用户的完整 UPN。可以通过设置 `kerberos.removeRealmFromPrincipal` 和 / 或 `kerberos.removeHostFromPrincipal` 控制主体名规范，这两项分别表示在比较 ID 之前移除域和移除第二部分。

表6-6：ZooKeeper模式

模式	描述	ACL ID 格式
`world`	代表任意用户	`anyone`
`digest`	代表使用密码认证的用户	<*用户名*>:base64(sha1sum(<*用户名*>:<*密码*>))
`ip`	使用客户端 IP 作为身份标识	*<ip>[/<cidr>]*
`sasl`	代表使用 SASL 认证的用户（例如 Kerberos 用户）	<*主体*>

6.5 Oozie授权

Apache Oozie 的授权模型很简单，只有两个级别的账户：用户（**user**）和管理员用户（**admin user**）。用户具有如下权限：

- 所有作业的读权限
- 自身作业的写权限
- 对作业的基于单作业访问控制列表（用户和组列表）的写权限
- 管理操作的读权限

管理员用户具有如下权限：

- 所有作业的写权限
- 管理操作的写权限

可以设置 `oozie- site.xml` 文件中的如下参数，以启用 Oozie 授权：

```
<property>
  <name>oozie.service.AuthorizationService.security.enabled</name>
  <value>true</value>
</property>
<property>
  <name>oozie.service.AuthorizationService.admin.groups</name>
  <value>oozie-admins</value>
</property>
```

如果没有设置 `oozie.service.AuthorizationService.admin.groups` 参数，可以在 `adminusers.txt` 文件中指定管理员用户列表，每行一个：

```
oozie
alice
```

除了所有者和管理员用户拥有对作业的写权限，也可以使用针对作业的访问控制列表授予用户写权限。Oozie ACL 使用的语法与 Hadoop ACL 的一样（见表 6-1），提交作业时，通过工作流、协调器（coordinator）或 `job.properties` 包文件的 `oozie.job.acl` 属性进行设置。

6.6 HBase和Accumulo的授权

Apache HBase 和 Apache Accumulo 是基于 Google 的 BigTable 设计的，建立在 HDFS 和 ZooKeeper 之上的排序的、分布式的键 / 值存储。这两个系统有着类似的数据模型，也都是为在 HDFS 上实现随机化访问和更新负载而设计的。数据存储在行里，每行包含一个或多个列。和关系型数据库不同的是，每行包含的列可以是不同的。这方便在数据记录的模式不尽相同时，实现复杂的数据模型。每行都通过名为行 **id** 或行键的主键进行检索，每一行中，每个值又进一步通过列键和时间戳进行检索。行键、列键、时间戳以及它们所指向的数据合在一起称为单元（**cell**）。HBase 和 Accumulo 内部将数据作为排序的键值对进行存储，其中键由行 ID、列键和时间戳组成。列键又分为两部分：列家族（**column family**）和列限定符（**column qualifier**）。HBase 中，同一个列家族的所有列被存储在单独的磁盘文件中。而 Accumulo 中，不同的列家族可以被合并到本地组（**locality group**）。

一系列排序的行称为“表”。HBase 中，列家族的集合是要预先为每个表定义好的。而 Accumulo 允许用户动态创建新的列家族。两个系统中，列限定符都不需要预先定义，任意限定符都可以被插入任意行。表的逻辑组（类似于数据库或者关系型数据库系统中的模式）称为“命名空间”。HBase 和 Accumulo 都支持系统、命名空间和表级别的权限，可用的权限及其含义各不相同，下面先看一下 Accumulo 的权限模型。

6.6.1 系统、命名空间和表级授权

Accumulo 最高支持系统级权限。通常来说，系统权限是为 Accumulo root 用户或 Accumulo 管理员预留的。高级别设置的权限会被低级别的对象所继承。例如，如果具有系统权限 `CREATE_TABLE`，则可以在任何命名空间创建表，即使没有显式地具有在该命名空间创建表的权限。表 6-7 给出了一系列系统级权限和描述，以及对应的命名空间级权限。

你将在本节看到多处提及 Accumulo root 用户，这与 root 系统账户是两码事。Accumulo root 用户是在 Accumulo 初始化时自动创建的，该用户被授予所有系统级权限。root 用户不可能取消这些权限，以防无人能管理 Accumulo。

表6-7：Accumulo中的系统级权限

权限	描述	对应的命名空间权限
`System.GRANT`	向其他用户授予权限的权限；为 Accumulo root 用户预留	`Namespace.ALTER_NAMESPACE`
`System.CREATE_TABLE`	创建表的权限	`Namespace.CREATE_TABLE`
`System.DROP_TABLE`	删除表的权限	`Namespace.DROP_TABLE`

（续）

权限	描述	对应的命名空间权限
System.ALTER_TABLE	修改表的权限	Namespace.ALTER_TABLE
System.DROP_NAMESPACE	删除命名空间的权限	Namespace.DROP_NAMESPACE
System.ALTER_NAMESPACE	修改命名空间的权限	Namespace.ALTER_NAMESPACE
System.CREATE_USER	创建新用户的权限	N/A
System.DROP_USER	删除用户的权限	N/A
System.ALTER_USER	修改用户密码、权限和授权的权限	N/A
System.SYSTEM	对表或用户执行管理操作的权限	N/A
System.CREATE_NAMESPACE	创建新命名空间的权限	N/A

命名空间是表的逻辑集合，有助于管理表以及将管理功能委托给较小的用户组。假设市场部门需要一系列 Accumulo 表以支撑其一些应用，为了减轻 Accumulo 管理员的负担，可以创建一个 marketing 命名空间，然后将 GRANT、CREATE_TABLE、DROP_TABLE 和 ALTER_TABLE 权限授予 marketing 中的一个管理员。这会允许该部门无需系统级权限或等待 Accumulo 管理员操作，即可创建和管理它自己的表。很多命名空间级权限会被命名空间中的表继承，表 6-8 给出了一系列命名空间级权限和描述，以及对应的表级权限。

表6-8：Accumulo中的命名空间级权限

权限	描述	对应的表级空间
Namespace.READ	读（scan）命名空间中的表的权限	Table.READ
Namespace.WRITE	写（put/delete）命名空间中的表的权限	Table.WRITE
Namespace.GRANT	向命名空间中的表授予权限的权限	Table.GRANT
Namespace.BULK_IMPORT	向命名空间中的表批量导入数据的权限	Table.BULK_IMPORT
Namespace.ALTER_TABLE	对命名空间中的表设置属性的权限	Table.ALTER_TABLE
Namespace.DROP_TABLE	删除命名空间中的表的权限	Table.DROP_TABLE
Namespace.CREATE_TABLE	在命名空间中创建表的权限	N/A
Namespace.ALTER_NAMESPACE	设置命名空间属性的权限	N/A
Namespace.DROP_NAMESPACE	删除命名空间的权限	N/A

表级权限用于控制对单个表的粗粒度访问。表 6-9 给出了一系列表级权限及其描述。

表6-9：Accumulo中的表级权限

权限	描述
Table.READ	读取（scan）表的权限
Table.WRITE	写（put/delete）表的权限
Table.BULK_IMPORT	向表中批量导入数据的权限
Table.ALTER_TABLE	设置表的属性的权限
Table.GRANT	授予表权限的权限
Table.DROP_TABLE	删除表的权限

可以用 Accumulo 命令行管理系统、命名空间和表级权限。特别地，使用 `grant` 命令可以授予权限，使用 `revoke` 命令可以撤销权限。示例 6-18 使用 Accumulo 命令行管理权限。

示例 6-18　使用 Accumulo 命令行管理权限

```
root@cloudcat> userpermissions -u alice
System permissions:

Namespace permissions (accumulo): Namespace.READ

Table permissions (accumulo.metadata): Table.READ
Table permissions (accumulo.root): Table.READ
root@cloudcat> user alice
Enter password for user alice: *****
alice@cloudcat> table super_secret_squirrel
alice@cloudcat super_secret_squirrel> scan
2014-03-31 16:11:06,828 [shell.Shell] ERROR: java.lang.RuntimeException:
  org.apache.accumulo.core.client.AccumuloSecurityException: Error PERMISSION_DENIED
  for user alice on table super_secret_squirrel(ID:a) - User does not have permission
  to perform this action
alice@cloudcat super_secret_squirrel> user root
Enter password for user root: ******
root@cloudcat super_secret_squirrel> grant Namespace.READ -ns "" -u alice
root@cloudcat super_secret_squirrel> user alice
Enter password for user alice: *****
alice@cloudcat super_secret_squirrel> scan
r f:c []    value
alice@cloudcat super_secret_squirrel>
```

HBase 在系统、命名空间和表级使用相同的 ACL 权限集（表 6-10）。高级别设置的权限会被低级别的对象继承。例如，如果授予一个用户系统级 READ 权限，则该用户可以读取集群的所有表。除了向单个用户授予权限之外，HBase 还支持向用户组授予权限。使用 grant 命令时，在组名前加上前缀 `@`，以分配组权限。HBase 使用与 Hadoop 中相同的用户到组的映射类，组映射默认情况下会加载 HBase Master 上的 Linux 组，也支持 LDAP 组或自定义映射。

表6-10：HBase中的权限

权限	描述
READ(R)	读（scan）权限
WRITE(W)	写（put/delete）权限
EXEC(X)	执行协处理器终端的权限
CREATE(C)	删除表、修改表属性以及添加、修改、删除列家族的权限
ADMIN(A)	启用和禁用表、触发区域重分配和迁移的权限以及通过 CREATE 授予的权限

示例 6-19 描述了使用系统级权限授予对所有表的读权限。首先 Alice 打开 HBase 命令行，获取表的列表，然后尝试读取 `super_secret_squirrel` 表。

示例 6-19　Alice 被拒绝访问一个 HBase 表

```
[alice@cdh5-hbase ~]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.0, rUnknown, Fri Feb  7 12:26:17 PST 2014

hbase(main):001:0> list
TABLE
super_secret_squirrel
1 row(s) in 2.2110 seconds
hbase(main):002:0> scan 'super_secret_squirrel'
ROW                                               COLUMN+CELL

ERROR: org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient
  permissions for user 'alice' for scanner open on table super_secret_squirrel

hbase(main):003:0> user_permission
User                                          Table,Family,Qualifier:Permission
0 row(s) in 0.7350 seconds
```

要注意的是，Alice 执行 `user_permission` 命令时，她看不到任何人。Alice 要求 HBase 管理员授予其访问 HBase 中所有表的权限，管理员以 hbase 用户的身份登录到 HBase 命令行，然后使用 `grant` 命令授予 Alice 系统级的 READ 权限。

示例 6-20　HBase 管理员授予 Alice 系统级 READ 权限

```
[hbase@cdh5-hbase ~]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.96.1.1-cdh5.0.0-beta-2, rUnknown, Fri Feb  7 12:26:17 PST 2014

hbase(main):001:0> grant 'alice', 'R'
0 row(s) in 2.6990 seconds

hbase(main):002:0> user_permission 'super_secret_squirrel'
User                                         Table,Family,Qualifier:Permission
 hbase                                       super_secret_squirrel,,:[Permission:
 actions=READ,WRITE,EXEC,CREATE,ADMIN]
1 row(s) in 0.2140 seconds
```

注意，由于 Alice 被授予的是系统级的权限，所以她仍然没有针对 `super_secret_squirrel` 表的权限。系统级的权限作为应用到 `hbase:acl` 表的内容，被展示在命令行里，如示例 6-21 所示。现在，当 Alice 执行 scan 动作，她就会得到表的行信息。

示例 6-21　Alice 现在可以扫描任意 HBase 表

```
hbase(main):004:0> user_permission
User                                          Table,Family,Qualifier:Permission
 alice                                        hbase:acl,,: [Permission: actions=READ]
1 row(s) in 0.1540 seconds

hbase(main):005:0> scan 'super_secret_squirrel'
ROW                                               COLUMN+CELL
```

```
 r                                                column=f:q, timestamp=1396369612376,
value=value
1 row(s) in 0.1310 seconds
```

6.6.2　列级别和单元级别授权

HBase 和 Accumulo 也支持数据级别的细粒度授权。HBase 中，可以在列级别指定权限。只有 `READ` 和 `WRITE` 权限可以应用到列级别 ACL。由于 HBase 支持分配权限到用户组，因此这是一种基于角色的访问控制（角色被一一映射到组）。HBase 模型与 Sentry 类似，也使用 RBAC，权限被存储在元数据层，而且在用户尝试访问表或列时得到应用。Accumulo 没有使用 RBAC 方式，而使用一种基于属性的安全。基于属性的安全使用标签对数据进行标记，并通过比较标签与用户的授权决定该用户是否具有读取数据值的权限。

Accumulo 中，安全标签存储在单元级，每个键值对都具有自己的标签。Accumulo 向 BigTable 的数据模型增加一个位于列限定符和时间戳中间的 `visibility` 元素，从而将安全标签作为键的一部分进行存储。和 Accumulo 中键的所有元素一样，安全标签不需要预先定义，可以在数据被插入时动态创建。为了支持更复杂的权限组合，安全标签包含一组使用布尔操作符 | 和 & 组合起来的自定义令牌，也可以使用括号制定布尔操作符的优先级。

除了和数据一起存储的标签之外，每个 Accumulo 用户还有一组安全标签。这些标签扫描数据时会与布尔表达式进行比较，然后过滤用户没有授权查看的单元。由于标签存储在单元级并且组成了键的一部分，所以实现多级安全很简单：同一行和列键可以指向不同授权级别的数据。这对于收集多个部分组成的相关数据（数据记录的一部分对所有用户公开，但更多的敏感部分是受限的）的公司来说，是个很强大的功能。

6.7　小结

本章介绍了允许或拒绝访问集群中数据或服务的授权机制。设置权限和 ACL 以控制对数据和资源的访问，是 Hadoop 管理的基础。我们看到不同组件的授权控制有所不同，尤其看到了授权访问数据的组件（HDFS、HBase、Accumulo）和授权访问计算和资源的组件（MapReduce、YARN）之间的区别。

至此，我们在单组件基础上实施的独立控制方面对授权进行了处理。虽然这对于限定对单个组件的访问很有效，但它增加了复杂度和管理员的负担，管理员需要学习这些不同的控制方法。第 7 章将介绍随着 Apache Sentry（孵化中）的引入，授权控制是如何趋于统一的。

第7章

Apache Sentry（孵化中）

Hadoop 生态系统项目的整个生命周期中，安全授权以多种形式被添加进系统。对于系统管理员来说，实现和维持一个包含多个组件的通用授权系统变得越来越有挑战性。使问题更复杂的是，这些多样化组件的授权控制机制拥有不同等级的粒度和执行策略，这经常使管理员对于用户实际能在 Hadoop 环境中做什么（或不做什么）感到困惑。同其他问题一样，这些问题是提出 Apache Sentry（孵化中）的驱动力。

为了使管理员可以对用户能访问什么进行更灵活的控制，Sentry 项目（http://wiki.apache.org/incubator/SentryProposal）提出了对细粒度、*基于角色的访问控制*（**role-based access controls**，**RBAC**）机制的需求。通常，HDFS 的授权控制被限制为简单的 POXIS 权限和扩展 ACL。那么，类似于 Hive、Cloudera Impala、Solr、HBase 和其他这种在 HDFS 顶层工作的框架，是怎样实现的呢？ Sentry 的目标是为 Hadoop 生态系统组件用统一的方法实现授权机制，这样安全管理员能够轻松地对用户和组所能访问的内容进行控制，而无需知道每一个 Hadoop 组件具体的输入和输出。

7.1 Sentry概念

每一个使用 Sentry 作为授权机制的组件必须与 Sentry *绑定*，该绑定是组件将授权决策委托给 Sentry 的插件。为了执行授权决策，它会应用相关*模型*。例如，SQL 模型能够应用于 Hive 和 Impala，Search 模型能够用于 Solr，BigTable 模型能用于 HBase 和 Accumulo。稍后将会详细讨论 Sentry 的特权模型。

通过引用恰当的模型，Sentry 利用*策略引擎*访问*策略提供者*，以决定请求的行为是否获得了授权。策略提供者是策略的存储机制，例如数据库或文本文件。图 7-1 为 Sentry 组件的概念性视图。

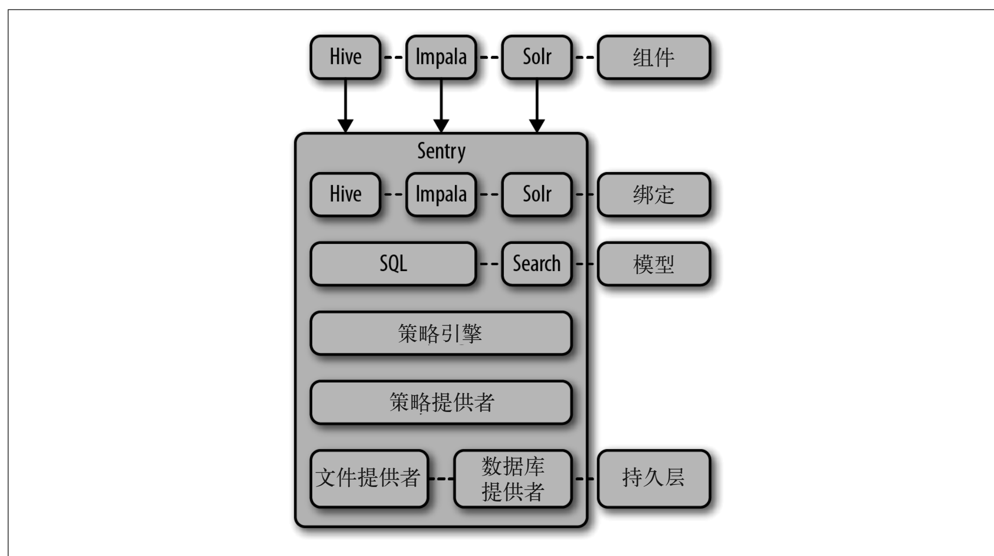


图 7-1：Sentry 组件

这个流程解释了组件如何从高层使用 Sentry，但策略引擎实际上的授权决策过程是怎样的呢？无论给定组件使用的模型如何，都有一些通用的关键概念。*用户*正是你所期望的样子，他们是执行某一特定行为的身份个体，这些行为可能是执行 SQL 查询、搜索某一集合、读取文件，或是检索一个键值对。用户也属于*组*。Sentry 的上下文中，组是拥有同样需求和特权的用户集合。Sentry 中的*特权*是一个数据访问单元，表示为一个包含对象和对该对象操作的元组。例如，对象可以是*数据库*、*表*或*集合*，操作可以是*创建*、*读*和*写*。



Sentry 特权通常以肯定的方式进行定义，因为默认情况下，Sentry 会拒绝对每一个对象的访问。这种特权不能与稍后讲到的 `REVOKE` 语法混淆，REVOKE 功能仅仅是移除肯定特权。

最后，*角色*是特权的集合，是 Sentry 中准予授权的基本单元。一个角色通常匹配一个业务功能，例如市场分析员或数据库管理员。用户、组、特权和角色之间的关系在 Sentry 中很重要，并且与下列逻辑紧密相关：

- 含有多个用户的组
- 被分配至一个组的角色
- 被授予特权的角色

如图 7-2 所示。

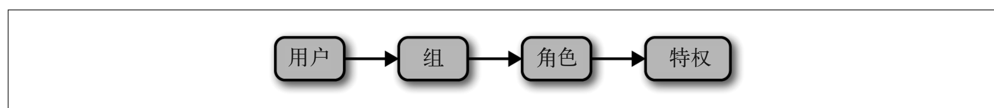


图 7-2：Sentry 实体间的关系

这种关系在 Sentry 中是严格强制执行的。例如，我们不可能将角色分配给一个用户，或是将特权授予一个组。该关系为严格执行时，会产生几种多对多关系。一个用户能够属于多个组，并且一个组可以包含多个用户。例如，Alice 可以同时隶属于市场组和开发组，并且开发组能够包含 Alice 和 Bob。

同样，一个角色能被分配给多个组，并且一个组能包含多个角色。例如，SQL 分析员角色能够同时被分配给市场组和开发组，并且开发组能够包含 SQL 分析员和数据库管理员两种角色。

最后，一个角色能够被授予多个特权，且某个特权能够成为多个角色的一部分。例如，SQL 分析员角色可以拥有点击流表中的 SELECT 特权和市场数据库中的 CREATE 特权，且同样的市场数据库中的 CREATE 特权也可以同时被赋给数据库管理员的角色。

了解 Sentry 高层概念后，下面继续深入了解实现环节。先从最新和最重要的开始：Sentry 服务。

7.2 Sentry服务

Sentry 项目首次进入 Apache 培育计划时，发布的第一个公开版本利用了基于插件的方法。能够利用 Sentry 的服务被配置 Sentry 插件（绑定模块），并且该插件运行于被配置的服务内部，直接读取策略文件。与许多 Hadoop 生态系统组件不同，Sentry 没有守护进程。此外，Sentry 策略在纯文本文件中进行配置，该文件枚举每一个策略。每当添加、修改或删除一个策略时，都需要对文件进行修改。可以想见，这种方法维护起来相当简朴和笨拙，并且很容易出错。使问题更严重的是，策略文件中的错误会使整个文件都失效！

幸运的是，Sentry 已经极大地超越了早期的这种方法，并且在 Hadoop 生态系统中已经成长为“一等公民”。从版本 1.4 开始，Sentry 配备了一个能够被 Hive 和 Impala 利用的服务。该服务使用一个数据库后端代替了文本文件，以存储策略。另外，使用 Sentry 的服务被配置了指向 Sentry 的绑定，而不是采用在本地处理所有授权决策的绑定。由于 Sentry 架构的进步，除了老旧的系统版本之外，不推荐 Hive 和 Impala 采用基于策略文件的配置。即使这样，本章还是会对两种配置的相关内容都进行讲解。图 7-3 描述了 Sentry 服务是怎样融入 SQL 访问的。

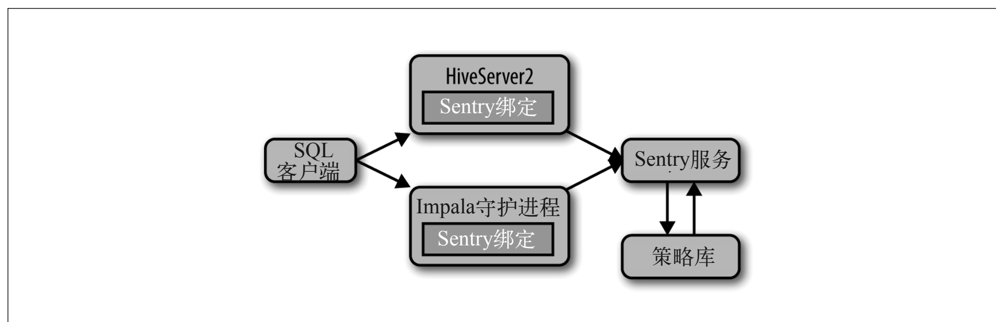


图 7-3：Sentry 服务架构

写作本书时，Solr 仍然在使用策略文件。可以预见，Solr 将会和其他新的启用 Sentry 的服务一样，抛弃基于策略文件的配置。

Sentry服务配置

配置 Sentry 并且在集群中运行的第一步是，要配置 Sentry 服务。Sentry 的主配置文件是 `sentry-site.xml`。示例 7-1 展示了在开启 Kerberos 的集群中对 Sentry 服务的一种典型配置，表 7-1 列举了配置参数。本章后续部分将讲解 Hadoop 生态系统组件是如何将 Sentry 服务用于授权的。

示例 7-1　Sentry 服务 sentry-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <property>
    <name>sentry.service.server.rpc-address</name>
    <value>server1.example.com</value>
  </property>
  <property>
    <name>sentry.service.server.rpc-port</name>
    <value>8038</value>
  </property>
  <property>
    <name>sentry.service.admin.group</name>
    <value>hive,impala,hue</value>
  </property>
  <property>
    <name>sentry.service.allow.connect</name>
    <value>hive,impala,hue</value>
  </property>
  <property>
    <name>sentry.store.group.mapping</name>
    <value>org.apache.sentry.provider.common.HadoopGroupMappingService</value>
  </property>
  <property>
    <name>sentry.service.server.principal</name>
    <value>sentry/_HOST@EXAMPLE.COM</value>
  </property>
  <property>
    <name>sentry.service.security.mode</name>
    <value>kerberos</value>
  </property>
  <property>
    <name>sentry.service.server.keytab</name>
    <value>sentry.keytab</value>
  </property>
  <property>
    <name>sentry.store.jdbc.url</name>
    <value>jdbc:mysql://server2.example.com:3306/</value>
  </property>
  <property>
    <name>sentry.store.jdbc.driver</name>
```

```
      <value>com.mysql.jdbc.Driver</value>
    </property>
    <property>
      <name>sentry.store.jdbc.user</name>
      <value>sentry</value>
    </property>
    <property>
      <name>sentry.store.jdbc.password</name>
      <value>sentry_password</value>
    </property>
  </configuration>
```

表 7-1 展示了 sentry-site.xml 相关的所有配置参数，包括用于配置 Sentry 服务的参数、用于配置基于策略文件实现的参数，以及组件特定配置参数。

表7-1：sentry-site.xml配置

配置	描述
hive.sentry.provider	对于 Hadoop 组来说通常为 org.apache.sentry.provider.file.HadoopGroupRe sourceAuthorizationProvider； 本地组的该参数仅能在策略文件部署中被定义为 org.apache.sentry.provider.file.LocalGro-upResourceAuthorizationProvider
hive.sentry.provider.resource	策略文件位置；可以为 file:// 和 hdfs://URIs
hive.sentry.server	Sentry 服务的名称；可以为任意名称
sentry.hive.provider.backend	Sentry 服务类型：org.apache.sentry.provider.file.SimpleFile-ProviderBackend 或者 org.apache.sentry.provider.db.Simple-DBProviderBackend
sentry.metastore.service.users	针对 Hive 的 metastore，能够绕过 Sentry 策略的用户列表；只适用于 Sentry 服务部署。
sentry.provider	与 hive.sentry.provider 相同的选项；用于 Solr
sentry.service.admin.group	用逗号间隔的组列表，是 Sentry 服务器的管理员
sentry.service.allow.connect	用逗号间隔的允许连接的用户列表；通常只是服务用户，不是终端用户
sentry.service.client.server.rpcaddress	Sentry 服务终端的客户端配置
sentry.service.client.server.rpcport	Sentry 服务端口的客户端配置
sentry.service.security.mode	Sentry 服务器正在处于的运行模式；可选 Kerberos 或无。
sentry.service.server.keytab	包含 sentry.service.server.principal 证书的 keytab 文件名
sentry.service.server.principal	sentry.service.server.keytab 包含的服务主体名，用于 Sentry 服务的身份标识
sentry.service.server.rpc-address	启动 Sentry 服务的主机名
sentry.service.server.rpc-port	监听的端口
sentry.solr.provider.resource	Solr 中策略文件的位置；既可以为 file://，也可以为 hdsf://URIs
sentry.store.jdbc.driver	用于连接数据库的 JDBC 驱动名
sentry.store.jdbc.password	连接时使用的 JDBC 口令

（续）

配置	描述
sentry.store.jdbc.url	Sentry 服务器连接后端数据库时使用的 JDBC URL
sentry.store.jdbc.user	连接时使用的 JDBC 用户名
sentry.store.group.mapping	提供用户到组的映射的类；通常为 org.apache.sentry.provider.common.HadoopGroupMappingService

7.3 Hive授权

Sentry 的典型实现是，向 Hive 加入基于角色的访问控制。没有强授权控制机制，Hive 用户可以在没有太多限制的情况下对 Hive 的 metastore 进行变更。另外，对 Hive 表的访问完全受底层 HDFS 文件系统访问许可的控制，该访问许可非常受限。通过 Sentry，安全管理员能够非常细粒度地控制哪些用户和组能使用 Hive，包括创建表和视图、向已存在的表插入新数据或查询数据等操作。

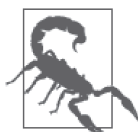
为了理解 Sentry 是如何向 Hive 提供授权机制的，首先需要理解 Hive 组件是如何合作的。Hive 架构中有 3 个主要组件：metastore 数据库、Hive Metastore Server 和 HiveServer2。metastore 数据库是一个关系型数据库，包括数据库、表和视图信息、表数据在 HDFS 中的位置、列的数据类型、文件格式和压缩等 Hive 元数据。当一个客户端与 Hive 互动时，这些信息对于理解即将执行的操作是十分必要的。

Hive 的早期版本中，这些差不多是用户能拥有的全部；Hive 客户端 API 会直接与 metastore 数据库对话并执行操作。从安全角度来说，这是件坏事。这个模型意味着每一个 Hive 客户端都拥有访问 Hive 元数据库的全部权限！ Hive Metastore Server 成为 Hive 架构下解决这个问题的组件。该角色的目的是成为 Hive 客户端和 metastore 数据库之间的中间层。通过这个模型，客户端只需要知道怎样与 Hive Metastore Server 联系，只有 Hive Metastore Server 负责与下层 metastore 数据库交互。

Hive 架构中的最后一个组件是 HiveServer2。这个组件的目的是通过 JDBC 和 ODBC 这类接口，向外部应用提供查询服务。HiveServer2 从客户端接收请求，与 Hive Metastore Server 通信，以取回元数据信息，并执行恰当的 Hive 操作，例如分发 MapReduce 作业。顾名思义，HiveServer2 是这种服务的第二个版本——第一版缺乏并发性和安全特性。理解这部分的重要之处在于，HiveServer2 最初就是用于向外部应用提供服务的。Hive 命令行接口（CLI）仍然与 Hive Metastore Server 直接交互，并使用 Hive API 执行操作。用户可以在 HiveServer2、`beeline` 中使用 CLI 执行操作，但这不是必须的。这个事实向面向所有客户端的强制授权机制提出了挑战。你可能猜到了，为了实现这个目标（强制授权），就要加强 HiveServer2 的安全授权，并确保所有 SQL 客户端都必须通过 HiveServer2 执行任何 Hive 的 SQL 操作。

Hive 架构中的另一个组件是 HCatalog。这是一个库的集合，该集合中的库能够允许非 SQL 客户端访问 Hive Metastore 结构体。这对于 Pig 或 MapReduce 用户很有用，他们可以不使用传统的 Hive 客户端确定文件的 metadata 结构体。HCatalog 库的一个扩展是 WebHCatServer 组件，该组件是一个提供 REST 接口以执行 HCatalog 函数的守护进程。

HCatalog 库和 WebHCatServer 都不利用 HiveServer2。所有通信都是直接与 Hive Metastore Server 进行的。由于这一点，Hive Metastore Server 也必须受到 Sentry 的保护，以确保 HCatalog 用户不能对 Hive Metastore 数据库进行任意修改。



虽然 Sentry 的 1.4 版能够对 Hive Metastore Server 提供写保护，但它并不同时限制读操作。这意味着，某个用户在 HCatalog 中进行一些相当于 `SHOW TABLES` 的操作时，系统会返回所有表的列表，包括用户没有权限访问的那些表。这与通过 HiveServer2 执行的相同操作不同，HiveServer2 中，用户只能看见他们有权限访问的那些对象。然而这仅仅是元数据泄漏。实际数据的访问在 HDFS 访问的时候仍旧会被强制控制。如果集群没有任何使用 HCatalog 的用户，那么对所有 Hive 和 HiveServer2 的流量实现强制访问控制的方法是，在配置文件 `core-site.xml` 中将属性 `hadoop.proxyuser.hive.groups` 设置为 `hive`，`impala`，这样就能允许有且仅有 Hive（HiveServer2）和 Impala（Catalog Server）都能直接访问 Hive Metastore Server。

图 7-4 展示了 Hive 架构的布局和 Sentry 的强制访问机制的发生位置。无论什么访问方法，核心的强制机制是保护 Hive 元数据不受非授权修改。

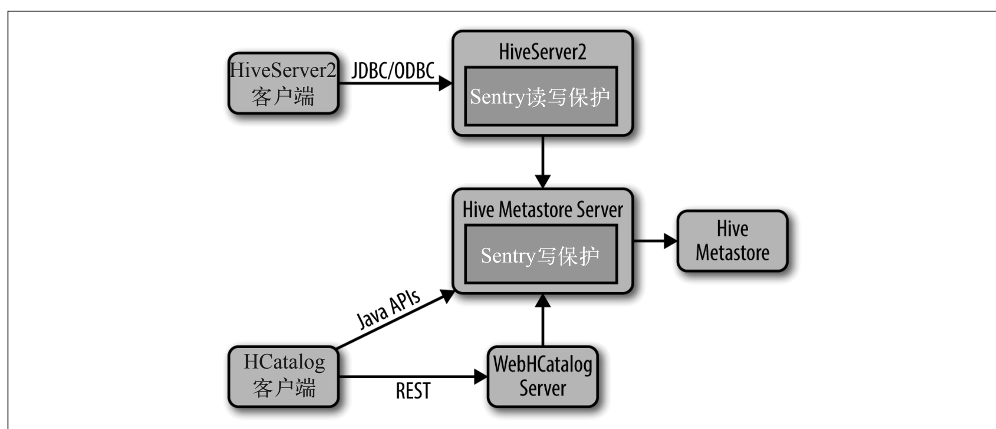


图 7-4：Hive sentry 架构

Hive Sentry的配置

本节将了解通过配置 Hive 使用 Sentry 授权的必需要素。示例 7-2 展示了能够利用 Sentry 授权的，用于 Hive Metastore Server 和 HiveServer2 的配置文件 `sentry-site.xml`。

示例 7-2　Hive sentry-site.xml 服务部署

```
<configuration>
  <property>
    <name>hive.sentry.server</name>
    <value>server1</value>
```

```
    </property>
    <property>
      <name>sentry.service.server.principal</name>
      <value>sentry/_HOST@EXAMPLE.COM</value>
    </property>
    <property>
      <name>sentry.service.security.mode</name>
      <value>kerberos</value>
    </property>
    <property>
      <name>sentry.hive.provider.backend</name>
      <value>org.apache.sentry.provider.db.SimpleDBProviderBackend</value>
    </property>
    <property>
      <name>sentry.service.client.server.rpc-address</name>
      <value>server1.example.com</value>
    </property>
    <property>
      <name>sentry.service.client.server.rpc-port</name>
      <value>8038</value>
    </property>
    <property>
      <name>hive.sentry.provider</name>
      <value>
         org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider
      </value>
    </property>
    <property>
      <name>sentry.metastore.service.users</name>
      <value>hive,impala,hue,hdfs</value>
    </property>
  </configuration>
```

示例 7-3 展示了使用 HiveServer2（与 Hive Metastore Server）和基于策略文件的部署时，Sentry 的一种典型配置。Sentry 中，基于策略文件的配置比基于服务的配置精简许多，但它们之间仍然存在共同点。稍后将看到，HiveServer2 守护进程的 `hive-site.xml` 配置文件中，指明了 `sentry-site.xml` 文件在本地文件系统中的位置。

示例 7-3　Hive sentry-site.xml 策略文件部署

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <property>
    <name>hive.sentry.provider</name>
    <value>
      org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider
    </value>
  </property>
  <property>
    <name>sentry.hive.provider.backend</name>
    <value>org.apache.sentry.provider.file.SimpleFileProviderBackend</value>
  </property>
  <property>
```

```
    <name>hive.sentry.provider.resource</name>
    <value>/user/hive/sentry/sentry-provider.ini</value>
  </property>
  <property>
    <name>hive.sentry.server</name>
    <value>server1</value>
  </property>
</configuration>
```

首先看看两个例子中都用到的配置属性。`hive.sentry.server` 参数为这个特定的 Sentry 服务器指定了名称（标签），可以在策略中引用。这个名称与机器的主机名无关。配置 `sentry.hive.provider.backend` 可以告诉 Hive 使用哪个后端服务。参数 `org.apache.sentry.provider.db.SimpleDBProviderBackend`（Sentry 服务）和 `org.apache.sentry.provider.file.SimplefileProviderBackend`. `hive.sentry.provider`（Sentry 策略文件）分别配置了 Sentry 确定组信息的方法。这里展示的 `HadoopGroupResourceAuthorizationProvider` 将会利用 hadoop 的所有配置方法，如从本地操作系统读取组，或者从 LDAP 上直接拉取组信息等。然而这仅是示例 7-2 举出的一种形式，因为 Sentry 服务不能使用策略文件定义本地用户到组的映射。

接下来了解专门针对 Sentry 服务的配置示例。Hive 如何连接到 Sentry 服务的细节由以下参数提供：

- `sentry.service.client.server.rpc-address`
- `sentry.service.client.server.rpc-port`

`sentry.service.server.principal` 和 `sentry.service.security.mode` 都会对 Kerberos 配置的细节进行设置。最后，`sentry.metastore.service.users` 的配置会列出那些被允许能够绕过 Sentry 授权机制，并且与 Hive Metastore Server 直连的用户——通常是类似于 Hive 和 Impala 的服务 / 系统用户。

对 `hive.sentry.provider.resource` 的配置是特定于域策略文件部署示例的。该参数指定了策略文件的存储位置。指定的 Sentry 策略文件位置会假定与 `hdfs-site.xml` 文件中指定的位置一样。例如，若 `hdfs-site.xml` 指向 HDFS，就会默认 HDFS 中存在路径 `user/hive/sentry/sentry-provider.ini`。此外也可以通过提供 HDFS 路径 `hdfs://` 或者本地文件系统路径 `file://` 明确存储位置。

`sentry-site.xml` 的配置对 Hive 很重要，与之形成对比的是，该文件并未对其本身启用 Sentry 授权机制。因此，在 Hive 的配置文件 `hive-site.xml` 中进行额外配置是必要的。示例 7-4 展示了 Hive Metastore Server 需要的相关配置，与之类似，示例 7-5 展示了 HiveServer2 所需的相关配置。这两种配置虽然相似，但仍有所区别。示例 7-6 中，最后的 `hive-site.xml` 示例展示了 HiveServer2 在基于策略文件的部署中所需的配置。注意，基于策略文件的部署中，Hive Metastore Server 不需要额外配置（稍后详述）。

示例 7-4　Hive Metastore Server hive-site.xml 的 Sentry 服务配置

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
```

```
  <!-- Unrelated properties omitted -->
  <property>
    <name>hive.sentry.conf.url</name>
    <value>file:///etc/hive/conf/sentry-site.xml</value>
  </property>
  <property>
    <name>hive.metastore.pre.event.listeners</name>
    <value>org.apache.sentry.binding.metastore.MetastoreAuthzBinding</value>
  </property>
  <property>
    <name>hive.metastore.event.listeners</name>
    <value>
      org.apache.sentry.binding.metastore.SentryMetastorePostEventListener
    </value>
  </property>
</configuration>
```

示例 7-5　HiveServer2 hive-site.xml 的 Sentry 服务配置

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <!-- Unrelated properties omitted -->
  <property>
    <name>hive.server2.enable.doAs</name>
    <value>false</value>
  </property>
  <property>
    <name>hive.server2.session.hook</name>
    <value>org.apache.sentry.binding.hive.HiveAuthzBindingSessionHook</value>
  </property>
  <property>
    <name>hive.sentry.conf.url</name>
    <value>file:///etc/hive/conf/sentry-site.xml</value>
  </property>
  <property>
    <name>hive.security.authorization.task.factory</name>
    <value>
      org.apache.sentry.binding.hive.SentryHiveAuthorizationTaskFactoryImpl
    </value>
  </property>
</configuration>
```

示例 7-6　HiveServer2 hive-site.xml 的 Sentry 策略文件配置

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <!-- Unrelated properties omitted -->
  <property>
    <name>hive.server2.enable.doAs</name>
    <value>false</value>
  </property>
  <property>
    <name>hive.server2.session.hook</name>
```

```
    <value>org.apache.sentry.binding.hive.HiveAuthzBindingSessionHook</value>
  </property>
  <property>
    <name>hive.sentry.conf.url</name>
    <value>file:///etc/hive/conf/sentry-site.xml</value>
  </property>
</configuration>
```

所有三个 hive-site.xml 示例中，配置参数 hive.sentry.conf.url 告诉 Hive 应到何处定位 Sentry 配置文件。HiveServer2 的两个示例中，参数 hive.server2.session.hook 用于指定一个绑定，该绑定会实质上将授权决策传递给 Sentry。

同样，HiveServer2 的两个示例中，要注意用户模拟功能是关闭的。关闭 Hive 的用户模拟是 Sentry 配置中的关键一环。为了真正实现从查询到数据访问这一过程的强制授权，Sentry 和 Hive 需要对查询接口和文件访问都进行控制。为此，Hive warehouse 的 HDFS 权限需要被锁定，如示例 7-7 所示。

示例 7-7　锁定 Hive 数据仓库

```
[alice@server1 ~]$ kinit hdfs
Enter password for hdfs@EXAMPLE.COM:
[alice@server1 ~]$ hdfs dfs -chown -R hive:hive /user/hive/warehouse
[alice@server1 ~]$ hdfs dfs -chmod -R 0771 /user/hive/warehouse
[alice@server1 ~]$
```

锁定 Hive warehouse 并且禁用用户模拟后，Sentry 便控制了查询接口的授权。由于只有 Hive 系统用户能够访问文件，因此 HDFS 权限也被锁定。从安全角度来说这样更好，而且这还使得 Sentry 拥有在低至视图级进行控制授权的能力。视图能被用于列级安全（只选择某些特定列）和行级安全，如提供一个 WHERE 子句的过滤器。如果启用用户模拟，那么查询行为会被当作终端用户。视图级权限没有被切合实际地强制限制，因为用户拥有 HDFS 的文件级（如表级）访问权限，并且能够通过直接访问文件绕过 Sentry 策略，例如 MapReduce。

7.4　Impala授权

Sentry 的初始版本包括对 Hive 和 Impala 的支持。虽然这两个组件拥有一些相似性，但它们在架构上有着本质区别，所以彻底理解 Sentry 如何融入系统之前，需要先对此进行强调。首先，Impala 是一整个处理框架。与 Hive 不同的地方在于，Hive 没有任何处理能力进行用户要求的实际工作。这种工作默认由 MapReduce 处理的（独立的版本 1，或者 YARN 上的版本 2）。

Impala 架构由三部分组成：Daemon、StateStore 和 Catalog 服务。Impala Daemon 即 impalad，是实际上的工作进程，它运行于集群中每一个运行有 HDFS DataNode 守护进程的节点。Impala StateStore 即 statestored，负责记录集群中所有 impalad 实例的健康状态。若一个实例出现问题，statestored 会将这个信息广播给其余所有 impalad 实例。虽然这看起来像是 Impala 架构中的核心组件，但实际上它不是必需的。如果 statestored 进程挂掉或根本不存在，那么所有被 impalad 实例完成的工作会继续运作。唯一潜在的影响是，如果一

个 impalad 实例变得不健康，则余下的实例会发现得很慢，这会导致总的查询执行时间延迟。Impala Catalog 服务又叫 `catalogd`，负责追踪记录元数据的变化。如果一个 Impala 查询在一个 impalad 上被执行，并以某种方式改变了元数据，catalogd 会将更新的元数据广播至其他 impalad 实例。catalogd 负责与 Hive Metastore Server 通信，以取得所有存在的元数据信息。

之前已经回顾了 Impala 架构的基础知识，下面学习 Sentry 真正发挥作用的地方。正如早先对 Hive 的讨论中阐述的那样，Sentry 是 Hive 组件 HiveServer2 和 Hive Metastore Server 的插件，大多数时候是集群中的独立实例。通过 Impala，Sentry 不再是用于扩充单一主组件的集中化插件，就像给 catalogd 或者 statestored 进程做的那样。实际上，Sentry 在每一个 impalad 实例中都被启用。用户通过 Sentry 连接某个特定 impalad 并进行一次查询时，impalad 利用 Sentry 策略（要么来自 Sentry 服务，要么来自策略文件）决定该用户是否被授权执行这次请求行为。

Impala的Sentry配置

正如前面学习 Hive 一样，本节将了解如何配置 Impala，使之利用 Sentry 进行授权控制。示例 7-8 展示了 Impala 守护进程为利用 Sentry 服务而使用的配置文件 `sentry-site.xml`。基于策略文件的部署中，`sentry-site.xml` 文件不是必需的。

示例 7-8　Impala sentry-site.xml 的服务部署

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <property>
    <name>sentry.service.server.principal</name>
    <value>sentry/_HOST@EXAMPLE.COM</value>
  </property>
  <property>
    <name>sentry.service.security.mode</name>
    <value>kerberos</value>
  </property>
  <property>
    <name>sentry.service.client.server.rpc-address</name>
    <value>server1.example.com</value>
  </property>
  <property>
    <name>sentry.service.client.server.rpc-port</name>
    <value>8038</value>
  </property>
</configuration>
```

你可能已经意识到，Impala 使用 Sentry 服务时，对 `sentry-site.xml` 的配置正是 Hive 同类配置的子集。上一节已经讨论了这些属性，现在可以直接看看要想启用 Sentry 授权机制，应如何配置 Impala 守护进程。

Sentry 服务的部署中，Impala 守护进程仅需要配置三个标识。第一个标识是 `server_name`，它是 Sentry 服务器的标签。该标识与 `hive.sentry.server` 的配置参数相匹配。第二个标识

是 `sentry_config`，将 Impala 守护进程指向配置文件 `sentry-site.xml` 的存储位置。第三个标识是 `authorized_proxy_user_config`，用于指定某些用户作为其他用户的模拟，例如 hue 用户。示例 7-9 展示了这种配置。

示例 7-9　用于 Sentry 服务配置的 Impala 标识

```
...Other unrelated flags omitted for brevity
-server_name=server1
-sentry_config=/etc/impala/conf/sentry-site.xml
-authorized_proxy_user_config=hue=*
```

基于 Sentry 策略文件的部署中，Impala 守护进程不需要 `Sentry_Config` 标识，而配置为 `authorization_policy_file` 和 `authorization_policy_provider_class` 标识。这些标识分别指明了 Sentry 策略文件的位置和授权提供者类的位置。后者的配置属性已被设为 `hive.sentry provider`，可达到同样的目的，如示例 7-10 所示。

示例 7-10

```
...Other unrelated flags omitted for brevity
-server_name=server1
-authorized_proxy_user_config=hue=*
-authorization_policy_file=/user/hive/sentry/sentry-provider.ini
-authorization_policy_provider_class=\
  org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider
```

7.5　Solr授权

Solr 的授权始于集合。集合是访问的主要入口，集合之于 Sole 类似于数据库之于 SQL。Sentry 授权最开始是在集合层面定义特权的。Sentry 由此演变出文件级授权。文件级授权通过给每个文件标上特定的字段名实现，该字段名包括与在 Sentry 策略文件（稍后进行阐述）中定义的相关 Sentry 角色名对应的值。给文件打标签的步骤将会在数据导入时进行，因此，通过良好的角色名避免重新处理文件修改标签是很重要的。

Solr的Sentry配置

本节阐述如何给 Solr 配置 Sentry 授权。示例 7-11 展示了如何在配置文件 `sentry-site.xml` 中对 Solr 服务器进行配置。

示例 7-11　Solr 的 sentry-site.xml 配置文件部署

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <property>
    <name>sentry.provider</name>
    <value>
      org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider
    </value>
  </property>
  <property>
```

```
    <name>sentry.solr.provider.resource</name>
    <value>/user/solr/sentry/sentry-provider.ini</value>
  </property>
</configuration>
```

示例 7-11 展示的两个配置参数此时看起来非常相似，但这些参数名在 Solr 中仍然略有区别。配置参数 sentry.provider 与 Hive 中的 hive.sentry.provider、Impala 中的 authorization_policy_provider_class 标识的工作原理相似。配置参数 sentry.solr.provider.resource 指定了 Sentry 策略文件的位置。重申一下，策略文件既可以放在本地文件系统，也可以放在 HDFS 中。该文件需要对正在运行 Solr 服务的用户（如 solr 用户）可读。

为了给 Solr 服务配置 Sentry 授权，需要一些环境变量。这些参数既可以配置为环境变量，也可以配置为 /etc/default/solr 配置文件中的参数行。第一个参数是 SOLR_SENTRY_ENABLED。显然，该参数被置为 ture 时，启用 Sentry 授权机制。下一个参数是 SOLR_AUTHORIZATION_SUPERUSER。该参数用于定义拥有超级用户权限的用户，即 solr 用户。最后一个参数为 SOLR_AUTHORIZATION_SENTRY_SITE，该参数指定了配置文件 sentry-site.xml 的位置（稍后介绍）。示例 7-12 展示了这些配置。

示例 7-12　Sentry 策略文件使用中的 Solr 环境变量

```
...Other unrelated environment variables omitted for brevity
SOLR_SENTRY_ENABLED=true
SOLR_AUTHORIZATION_SUPERUSER=solr
SOLR_AUTHORIZATION_SENTRY_SITE=/etc/solr/conf/sentry-site.xml
```

本节早先提到过，能够进行文件级授权。为了实现这个功能，有必要对集合进行一些配置。集合的配置默认通过配置文件 solrconfig.xml 进行。该文件如示例 7-13 所示。

示例 7-13　文件级安全 solrconfig.xml

```
<searchComponent name="queryDocAuthorization"
class="org.apache.solr.handler.component.QueryDocAuthorizationComponent">
    <bool name="enabled">true</bool>
    <str name="sentryAuthField">sentry_auth</str>
    <str name="allRolesToken">*</str>
</searchComponent>
```

示例 7-13 展示了 org.apache.solr.handler.component.QueryDocAuthorizationComponent 类被用于文件级授权决策，将 enabled 参数设为 true 即可开启。配置参数 sentryAuthfield 定义了文件包含能确定访问权限的授权令牌的字段名称。senty_auth 的默认值如示例所示，但也可被设置为任意值。文件将会使用这个标签插入访问本文件所需的角色名。最后一个配置参数 allRolesToken 定义了一个允许所有角色都可访问某个文件的令牌。该参数默认值为 *，可以保持该值不变，从而与其他 Sentry 特权的通配符保持一致。

7.6　Sentry特权模型

本节将了解 Sentry 提供授权的多种服务的特权模型。特权模型能够标识特权、特权对应的对象类型、特权有效范围及其他有助于安全管理员理解可用授权控制的相关信息和粒度。

充分理解特权模型能够确保选择合适的策略，以满足授权控制需要的程度，并保护数据不会受到非授权的访问。

7.6.1 SQL特权模型

Sentry 为 SQL 访问提供了三种权限：SELECT、INSERT 和 ALL。这些权限并不可用于每一个对象。表 7-2 提供了 SQL 上下文中权限和对象的对应情况。SQL 权限模型是分层的，这意味着容器客体会将特权传递给子对象。这对于透彻理解用户是否拥有访问权限是很重要的。

表7-2：SQL特权类型[a]

特权	对象
INSERT	TABLE, URI
SELECT	TABLE, VIEW, URI
ALL	SERVER, DB, URI

a 所有特权模型表都由 Cloudera, Inc 授权，从 cloudera.com 复制而来。

表 7-3 列出了容器权限在给定对象处的受到粒度特权的支配。例如，表中第一行可理解为“一个 SERVER 对象的 ALL 权限使得 DATABASE 对象也拥有 ALL 权限”。

表7-3：SQL权限层级

基础对象	粒度权限	容器对象	容器对象传递的粒度权限
DATABASE	ALL	SERVER	ALL
TABLE	INSERT	DATABASE	ALL
TABLE	INSERT	DATABASE	ALL
VIEW	SELECT	DATABASE	ALL

SQL 特权模型的最后一部分是，理解权限如何映射到 SQL 操作。表 7-4 展示了某个给定的 SQL 操作、操作应用的对象范围和执行该操作需要什么权限。例如，表中第一行可理解为“CREATE DATABASE 操作可应用于 SERVER 对象并且需要对 SERVER 对象的 ALL 权限”。一些 SQL 操作需要不止一种权限，例如创建视图。创建一个新的视图需要有对视图所在 DATABASE 的 ALL 权限，也需要有对于被新视图引用的表 / 视图对象的 SELECT 权限。

表7-4：SQL权限

SQL操作	范围	权限
CREATE DATABASE	SERVER	ALL
DROP DATABASE	DATABASE	ALL
CREATE TABLE	DATABASE	ALL
DROP TABLE	TABLE	ALL

（续）

SQL操作	范围	权限
CREATE VIEW	DATABASE; SELECT on TABLE	ALL
DROP VIEW	VIEW/TABLE	ALL
CREATE INDEX	TABLE	ALL
DROP INDEX	TABLE	ALL
ALTER TABLE ADD COLUMNS	TABLE	ALL
ALTER TABLE REPALCE COLUMNS	TABLE	ALL
ALTER TABLE CHANGE column	TABLE	ALL
ALTER TABLE RENAME	TABLE	ALL
ALTER TABLE SET TBLPROPERTIES	TABLE	ALL
ALTER TABLE SET FILEFORMAT	TABLE	ALL
ALTER TABLER SET LOCATION	TABLE	ALL
ALTER TABLE ADD PARTITION	TABLE	ALL
ALTER TABLE ADD PARTITION location	TABLE	ALL
ALTER TABLE DROP PARTITION	TABLE	ALL
ALTER TABLE PARTITION SET FILEFORMAT	TABLE	ALL
SHOW TBLPROPERTIES	TABLE	SELECT/INSERT
SHOW CREATE TABLE	TABLE	SELECT/INSERT
SHOW PARTITIONS	TABLE	SELECT/INSERT
DESCRIBE TABLE	TABLE	SELECT/INSERT
DESCRIBE TABLE PARTITION	TABLE	SELECT/INSERT
LOAD DATA	TABLE; URI	INSERT
SELECT	TABLE	SELECT
INSERT OVERWRITE TABLE	TABLE	INSERT
CREATE TABLE AS SELECT	DATABASE; SELECT on TABLE	ALL
USE database	ANY	ANY
ALTER TABLE SET SERDEPROPERTIES	TABLE	ALL
ALTER TABLE PARTITION SET SERDEPROPERTIES	TABLE	ALL
CREATE ROLE	SERVER	ALL
GRANT ROLE TO GROUP	SERVER	ALL
GRANT PRIVILEGE ON SERVER	SERVER	ALL
GRANT PRIVILEGE ON DATABASE	DATABASE	WITH GRANT OPTION
GRANT PRIVILEGE ON TABLE	TABLE	WITH GRANT OPTION

虽然 Hive 和 Impala 支持大多数 SQL 操作，但仍有一些操作仅能被 Hive 或 Impala 一方支持，或均尚未实现。表 7-5 列出了仅能应用于 Hive 的 SQL 权限，表 7-6 列出了仅能应用于 Impala 的 SQL 权限。

表7-5：仅用于Hive的SQL权限

SQL操作	范围	权限
INSERT OVERWRITE DIRECTORY	TABLE; URI	INSERT
ANALYZE TABLE	TABLE	SELECT+INSERT
IMPORT TABLE	DATABASE; URI	ALL
EXPORT TABLE	TABLE; URI	SELECT
ALTER TABLE TOUCH	TABLE	ALL
ALTER TABLE TOUCH PARTITION	TABLE	ALL
ALTER TABLE CLUSTERED BY SORTED BY	TABLE	ALL
ALTER TABLE ENABLE/DISABLE	TABLE	ALL
ALTER TABLE PARTITION ENABLE/DISABLE	TABLE	ALL
ALTER TABLE PARTITION RENAME TO PARTITION	TABLE	ALL
ALTER DATABASE	DATABASE	ALL
DESCRIBE DATABASE	DATABASE	SELECT/INSERT
SHOW COLUMNS	TABLE	SELECT/INSERT
SHOW INDEXES	TABLE	SELECT/INSERT

表7-6：仅用于Impala的SQL权限

SQL操作	范围	权限
EXPLAIN	TABLE	SELECT
INVALIDATE METADATA	SERVER	ALL
INVALIDATE METADATA table	TABLE	SELECT/INSERT
REFRESH table	TABLE	SELECT/INSERT
CREATE FUNCTION	SERVER	ALL
DROP FUNCTION	SERVER	ALL
COMPUTE STATS	TABLE	ALL

7.6.2 Solr特权模型

对于 Solr，Sentry 提供三种权限：`QUERY`、`UPDATE` 和 *(ALL)。Solr 的特权模型分为应用于请求处理程序（handler）的特权和应用于集合的特权。表 7-8~ 表 7-10 中，`admin` 集合名在 Sentry 中是用于代表管理行为的特殊集合。所有 Solr 特权模型表中，`collection1` 表示一个任意集合名。

表7-7：非管理事务请求处理程序的Solr权限表

请求处理程序	所需权限	需要权限的集合
select	QUERY	collection1
query	QUERY	collection1
get	QUERY	collection1
browse	QUERY	collection1
tvrh	QUERY	collection1

（续）

请求处理程序	所需权限	需要权限的集合
clustering	QUERY	collection1
terms	QUERY	collection1
elevate	QUERY	collection1
analysis/field	QUERY	collection1
analysis/document	QUERY	collection1
update	UPDATE	collection1
update/json	UPDATE	collection1
update/csv	UPDATE	collection1

表7-8：admin集合操作的Solr权限表

集合操作	所需权限	需要权限的集合
create	UPDATE	admin, collection1
delete	UPDATE	admin, collection1
reload	UPDATE	admin, collection1
createAlias	UPDATE	admin, collection1
deleteAlias	UPDATE	admin, collection1
syncShard	UPDATE	admin, collection1
splitShard	UPDATE	admin, collection1
delelteShard	UPDATE	admin, collection1

表7-9：核心admin操作的Solr权限表

集合操作	所需权限	需要权限的集合
create	UPDATE	admin, collection1
rename	UPDATE	admin, collection1
load	UPDATE	admin, collection1
unload	UPDATE	admin, collection1
status	UPDATE	admin, collection1
persist	UPDATE	admin
reload	UPDATE	admin, collection1
swap	UPDATE	admin, collection1
mergeIndexes	UPDATE	admin, collection1
split	UPDATE	admin, collection1
prepRecover	UPDATE	admin, collection1
requestRecover	UPDATE	admin, collection1
requestSyncShard	UPDATE	admin, collection1
requestApplyUpdates	UPDATE	admin, collection1

表7-10：信息和管理员处理程序的Solr权限表

请求处理程序	所需权限	需要权限的集合
LukeRequestHandler	QUERY	admin
SystemInfoHandler	QUERY	admin
SolrInfoMBeanHandler	QUERY	admin
PluginInfoHandler	QUERY	admin
ThreadDumpHandler	QUERY	admin
PropertiesRequestHandler	QUERY	admin
LogginHandler	QUERY, UPDATE(or*)	admin
ShowFileRequestHandler	QUERY	admin

7.7 Sentry策略管理

前面已经讲解了权限表和可用的访问方式，下面进一步了解实际的策略是怎样被添加、删除或修改的。管理不同策略的方法要依赖于 Sentry 的部署类型，要么是新的 Sentry 服务，要么是旧的策略文件。对于前者（这也是我们推荐的类型），管理策略文件的方法是通过 SQL 命令实现的。

7.7.1 SQL命令

习惯了在流行的关系型数据库系统中管理角色和许可的安全管理员会发现，管理 Sentry 策略的 SQL 语法十分眼熟。表 7-11 展示了管理员管理 Sentry 策略时的所有可用声明。

表7-11：Sentry策略的SQL语法

声明	描述
`CREATE ROLE` *`role_name`*	创建拥有指定名称的角色
`DROP ROLE` *`role_name`*	删除拥有指定名称的角色
`GRANT ROLE` *`role_name`* `TO GROUP` *`group_name`*	分配指定角色到指定的组
`REVOKE ROLE` *`role_name`* `FROM GROUP` *`group_name`*	从指定的组移出指定角色
`GRANT privilege ON object TO ROLE` *`role_name`*	将对某指定对象的权限分配给指定角色
`GRANT privilege ON object TO ROLE` *`role_name`* `WITH GRANT OPTION`	将对某指定对象的权限分配给指定角色并允许该角色获取更多该对象的权限
`REVOKE privilege ON object FROM ROLE` *`role_name`*	取消指定角色的对指定对象的权限
`SET ROLE` *`role_name`*	为当前会话设置指定角色
`SET ROLE ALL`	为当前会话启用（用户能访问的）所有角色
`SET ROLE NONE`	禁用当前会话的所有角色
`SHOW ROLES`	列出数据库中的所有角色
`SHOW CURRENT ROLES`	显示当前会话启用的所有角色
`SHOW ROLE GRANT GROUP` *`group_name`*	显示指定组的所有角色
`SHOW GRANT ROLE` *`role_name`*	显示指定角色的所有许可权限
`SHOW GRANT ROLE` *`role_name`* `ON object` *`object_name`*	显示指定角色关于指定对象的所有许可权限

表 7-11 提供了多种语法的列表，一个处于工作状态中的示例能保证看到这些行为（示例 7-14）。

示例 7-14　Sentry SQL 的使用示例

```
# Authenticated as the hive user, which is a member of a group listed in
# sentry.service.admin.group and accessing HiveServer2
# via the beeline CLI

# Create the role for hive administrators
0: jdbc:hive2://server1.example.com:100> CREATE ROLE hive_admin;
No rows affected (0.852 seconds)

# Grant the hive administrator role to the sqladmin group
0: jdbc:hive2://server1.example.com:100> GRANT ROLE hive_admin TO GROUP sqladmin;
No rows affected (0.305 seconds)

# Grant server-wide permissions to the hive_admin role
0: jdbc:hive2://server1.example.com:100> GRANT ALL ON SERVER server1
TO ROLE hive_admin;
No rows affected (0.339 seconds)

# Show all of the roles in the Sentry database
0: jdbc:hive2://server1.example.com:100> SHOW ROLES;

+-------------+
|    role     |
+-------------+
| hive_admin  |
+-------------+
1 row selected (0.63 seconds)

# Show all the privileges that the hive_admin role has access to
# (some columns omitted for brevity)
0: jdbc:hive2://server1.example.com:100> SHOW GRANT ROLE hive_admin;
+-----------+----------------+----------------+------------+---------------+
| database  | principal_name | principal_type | privilege  | grant_option  |
+-----------+----------------+----------------+------------+---------------+
| *         | hive_admin     | ROLE           | *          | false         |
+-----------+----------------+----------------+------------+---------------+
+----------+
| grantor  |
+----------+
| hive     |
+----------+
1 row selected (0.5 seconds)

# Show all the roles that the sqladmin is a part of
0: jdbc:hive2://server1.example.com:100> SHOW ROLE GRANT GROUP sqladmin;
+-------------+--------------+------------+----------+
|    role     | grant_option | grant_time | grantor  |
+-------------+--------------+------------+----------+
| hive_admin  | false        |            | hive     |
+-------------+--------------+------------+----------+
1 row selected (0.5 seconds)
```

```
# Remove all of the roles for the current user session
0: jdbc:hive2://server1.example.com:100> SET ROLE NONE;
No rows affected (0.2 seconds)

# Show list of current roles
0: jdbc:hive2://server1.example.com> SHOW CURRENT ROLES;
+-------+
| role  |
+-------+
+-------+
No rows selected (0.305 seconds)

# Verify that no roles yields no access
0: jdbc:hive2://server1.example.com:100> SHOW TABLES;
+-----------+
| tab_name  |
+-----------+
+-----------+
0: jdbc:hive2://server1.example.com:100> SELECT COUNT(*) FROM sample_07;
Error: Error while compiling statement:
FAILED: SemanticException No valid privileges (state=42000,code=40000)

# Set the current role to the hive_admin role
0: jdbc:hive2://server1.example.com:100> SET ROLE hive_admin;
No rows affected (0.176 seconds)

# Show list of current roles
0: jdbc:hive2://server1.example.com:100> SHOW CURRENT ROLES;
+-------------+
|     role    |
+-------------+
| hive_admin  |
+-------------+
1 row selected (0.404 seconds)

# Execute commands that are permitted
0: jdbc:hive2://server1.example.com:100> SHOW TABLES;
+------------+
|  tab_name  |
+------------+
| sample_07  |
| sample_08  |
+------------+
2 rows selected (0.536 seconds)
0: jdbc:hive2://server1.example.com:100> SELECT COUNT(*) FROM sample_07;
+------+
| _c0  |
+------+
| 823  |
+------+
1 row selected (20.811 seconds)
```



使用 WITH GRANT OPTION 能大幅减轻全局 SQL 管理员的负担。通用的做法是，创建一个指定业务范围的 Hive 数据库，并将管理员权限分配给一个数据库专用的 admin 角色。这样指定业务范围后，就有一定灵活度管理它们拥有的数据的权限。要确定某角色是否有这个选项，可以用 SHOW GRANT ROLE role_name 命令查看列的 grant_option。

示例 7-14 中，命令由 HiveServer2 的 beeline CLI 执行，但它们也可以使用 impala-shell 运行。这两个组件使用了同样的 Sentry 服务，因而具有同样的 Sentry 策略，所以从其中一个组件进行的修改会立刻在另一个组件中反映出来。Sentry 授权决策不会被独立的组件缓存，这是从安全的后果考虑的。

7.7.2 SQL策略文件

使用为 SQL 组件提供 Sentry 服务的 Sentry 部署方式时，策略管理是熟悉且简洁易懂的，而使用传统的基于策略文件的实现则并非如此。启用 Sentry 的组件需要拥有对策略文件的读权限。在 Hive 和 Impala 中使用策略文件时，可以通过使 hive 组包含 file 组，并且确保 hive 和 impala 用户都是该组成员的方式实现。策略文件本身既可以放在本地文件系统，也可以放在 HDFS 中。对于前者，文件需要存储做出授权决策的组件被部署的地方。例如，Hive 启用 Sentry 时，本地策略文件需要存储在运行有 HiveServer2 守护进程的机器上。



为了利用 HDFS 同步复制机制带来的冗余和高可用性，强烈推荐在 HDFS 中指定一个位置存储策略文件。因为每一个用户操作都需要读取策略文件，明智的做法是增加该文件的副本，使得读取它的组件能够从许多不同的节点进行检索。这可以通过 hdfs dfs -setrep *N* /user/hive/sentry/sentry-provider.ini 实现，此处的 *N* 是所需的副本数。策略文件很小，所以将副本的数量设为集群中 DataNode 的数量是完全可行的。

策略文件的格式遵守典型的 ini 文件格式，带有被中括号标识的配置段和独立的键值对配置部分。示例 7-15 展示了适用于 Hive 和 Impala 的 Sentry 样例配置文件。

示例 7-15　SQL sentry-provider.ini

```
[databases]
product = hdfs://nameservice1/user/hive/sentry/product.ini

[groups]
admins = admin_role, tmp_access
analysts = analyst_role, tmp_access
developers = developer_role, tmp_access
etl = etl_role, tmp_access

[roles]
# uri accesses
tmp_access = server=server1->uri=hdfs://nameservice1/tmp

# default database accesses
```

```
analyst_role = server=server1->db=default->table=*->action=select
developer_role = server=server1->db=default
etl_role = server=server1->db=default->table=*->action=insert, \
           server=server1->db=default->table=*->action=select

# administrative role
admin_role = server=server1
```

示例 7-15 中的策略文件做了很多事，可能并不能一眼就看出它定义了什么。策略文件的第一段是*数据库*。本段列出了所有数据库，以及为了确保对它们的安全访问所使用的相应策略文件。每个数据库不会强制要求拥有隔离的配置文件。然而，将配置文件隔离能带来以下好处：

- 能够进行版本跟踪，以轻易区分哪个数据库受到策略变化的影响以及受影响的时间；
- 能以单个数据库的粒度进行授权管理控制；
- 某个数据库策略文件的错误配置不会影响主策略文件 `sentry-provider.ini` 或其他数据库策略文件；
- 能够通过改变 HDFS 中对策略文件的访问权轻松禁用对整个数据库的访问，而不用改动策略文件本身的内容。

策略文件的第二段是*组*。本段提供了组和它们被分配到的角色之间的映射，语法是 `group=role`。正如之前讨论过的，组来自两个地方：来源于 Hadoop 的组，或者是本地专门为 Sentry 配置的组。示例 7-15 中没有定义本地配置的组，因为早先示例 7-1 中的 `sentry-site.xml` 被配置为 `HadoopGroupResourceAuthorizationProvider`。以下是关于策略文件中组配置段的几个要素：

- 给定组能够分配给许多角色，用逗号隔开；
- 组配置段中的条目是从上向下读的，从而使得同组中的重复条目能够覆盖任何先前的条目；
- 组名都是全局的，无论它们是在本地定义的还是由 Hadoop 提供的；
- 角色名是本地的，分配给一个组的角色名仅适用于配置它的那个文件。

策略文件的最后一段是*角色*。这里定义安全策略的主要部分。角色配置的语法是 `role=permission`。配置的许可部分看起来有些奇怪，因为这部分也是 `key=value` 形式的语法，但通过每个键值对配置之间的箭头实现了粒度更精确的许可定义。这可以通过 `admin_role` 权限的定义证明。该角色被授予 **server** 级别的全部访问权限。下一粒度级别是数据库级，或 **db** 级。`developer_role` 的权限定义授予了访问默认数据库的全部权限。这之后的下一级别是 **table** 级。该例展示了策略文件的另一个特性——它支持使用通配符选项指代“任意”表。

通配符仅用于代表任意值。通过部分名称匹配引用数据库表时，它们不能用作引用的开头和结尾。这看似是对通配符实现的一种限制，但用户应牢记，这是在制定安全策略。使用通配符进行部分名称匹配有可能会意外地将特权授予非授权用户。想象一下这样的场景，宾夕法尼亚州的一组开发者用户被授权能够访问任何名称是 pa 的表，但稍后人力资源部的用户开始使用这个集群，并建立了一个名为 payroll 的表，该表包含所有公司雇员的工资单。现在，宾夕法尼亚的开发者组无意中能够访问保密信息了。因此，在安全策略中使用通配符时，应当格外小心。

操作行为属于权限最高级粒度的角色定义的上下文内。Hive 和 Impala 的表的上下文中，仅支持 `select` 和 `insert` 操作。这些操作是完全互斥的。如果一个角色对一个表既打算进行 select 操作，也打算进行 insert 操作，那么需要获得这两种操作的许可。组的配置段中，多项权限能够被分配给一个角色。为了实现这个目的，可以简单地将它们用逗号隔开。为了增加可读性，可使用反斜杠进行权限定义的换行连接，如示例 7-15 中的 `etl_role` 所示。

角色段的最后一部分讨论了 URI 的概念。示例 7-15 展示了 `tmp_access` 角色的一个 URI 访问许可。该许可允许用户做两件事：为本处数据创建外部表，从其有权限访问的其他本地表导出数据。



默认的 URI 访问只能在 `sentry-provider.ini` 中被指定，而不能在每个数据库各自的策略文件中被指定。进行限制的原因是，单独的管理员会维护一个数据库策略文件，但不会对其他策略文件也进行维护管理。如果管理员能够在策略文件中定义 URI 访问控制，他们就可以允许自己或其他任何人访问 HDFS 中对 `hive` 用户通过外部表可读的任意位置。如果这种行为需要被重写，则应当在 HiveServer2 中对 Java 配置选项 `sentry.allow.uri.db.policyfile=true` 进行配置。该配置应该仅能在所有管理员对所有 Sentry 策略文件都具有拥有同样的访问权的情况下使用。

7.7.3　Solr策略文件

Hive 和 Impala 可以通过 SQL 语法使用 Sentry 服务和管理策略，而 Solr 还在使用策略文件的方法。策略文件的格式类似于 SQL 中相应的部分，只是有几处变化。不同于 SQL 组件那样在数据库和表上操作，Solr 授权是在集合上操作的。同样地，Solr 的特权不含 SELECT 和 INSERT，而使用 Query 和 Update 代替。Solr 特权可以被设为 All，通过星号（*）实现。

示例 7-16 展示了一个与 SQL 示例相似的设计。在组配置段，组被赋值为角色；在角色配置段，角色被赋值为特权。`analyst_role` 提供了查询 `customer_logs` 集合的权限，`etl_role` 提供了更新它的权限，`developer_role` 提供了访问它的所有权限。最后，`admin_role` 拥有针对 admin 集合的所有权限。

示例 7-16　Solr sentry-provider.ini

```
[groups]
admins = admin_role
analysts = analyst_role
developers = developer_role
etl = etl_role

[roles]
analyst_role = collection=customer_logs->action=Query
developer_role = collection=customer_logs->action=*
etl_role = collection=customer_logs->action=Update

# administrative role
admin_role = collection=admin->action=*
```

应当指出，虽然 SQL 策略文件允许每个数据库拥有独立的策略文件，但 Solr 却不允许这样做。这意味着 Solr 策略管理员在修改策略时需要格外当心，因为就像 SQL 策略文件那样，一个语法错误会使整个策略文件失效，从而无意间导致所有数据库都拒绝访问。一个对付拼写错误和语法错误的好办法是，使用 `config-tool` 验证策略文件，这正是下一节要讲的。

7.7.4 策略文件的验证和校验

既然 Sentry 一开始被设计为使用纯文本格式的策略文件，那么管理员显然需要某种校验工具，在文件生效之前对其进行基本的合理性检查。Sentry 附带了一个名为 `sentry` 的二进制程序（这名称真是毫无新意），它为策略文件的实现提供了一个重要功能：`config-tool` 命令。该命令不仅允许管理员检查策略文件中的错误，还提供了一个针对给定用户的特权验证机制。示例 7-17 展示了对策略文件的校验，其中第一个策略文件没有错误，第二个有一处拼写错误（server 被误拼为 sever）。

示例 7-17　Sentry config-tool 校验

```
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool
-s file:///etc/sentry/sentry-site.xml -i file:///etc/sentry/sentry-provider.ini -v
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
No errors found in the policy file
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool
-s file:///etc/sentry/sentry-site.xml -i file:///etc/sentry/sentry-provider2.ini -v
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider2.ini
Sentry server: server1
 *** Found configuration problems ***
ERROR: Error processing file file:/etc/sentry/sentry-provider2.ini
 No authorizable found for sever=server1
ERROR: Failed to process global policy file
 file:/etc/sentry/sentry-provider2.ini
Sentry tool reported Errors:
org.apache.sentry.core.common.SentryConfigurationException:
        at org.apache.sentry.provider.file.SimpleFileProviderBackend.
         validatePolicy(SimpleFileProviderBackend.java:198)
        at org.apache.sentry.policy.db.SimpleDBPolicyEngine.
         validatePolicy(SimpleDBPolicyEngine.java:87)
        at org.apache.sentry.provider.common.ResourceAuthorizationProvider.
         validateResource(ResourceAuthorizationProvider.java:170)
        at org.apache.sentry.binding.hive.authz.SentryConfigTool.
         validatePolicy(SentryConfigTool.java:247)
        at org.apache.sentry.binding.hive.authz.
```

```
    SentryConfigTool$CommandImpl.run(SentryConfigTool.java:638)
        at org.apache.sentry.SentryMain.main(SentryMain.java:94)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.
         invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.
         invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:212
[root@server1 ~]#
```

config-tool 提供的另一个强大功能是验证用户的特权。该功能可以通过列出指定用户的所有权限实现，也可以更具体地通过测试指定用户能否被授权执行某次查询实现。示例 7-18 展示了这些功能的使用。

示例 7-18　Sentry config-tool 验证

```
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool \
 -s file:///etc/sentry/sentry-site.xml \
 -i file:///etc/sentry/sentry-provider.ini -l -u bob
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
Available privileges for user bob:
        server=server1
        server=server1->uri=hdfs://server1.example.com:8020/tmp
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool \
 -s file:///etc/sentry/sentry-site.xml \
 -i file:///etc/sentry/sentry-provider.ini -l -u alice
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
Available privileges for user alice:
        *** No permissions available ***
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool
 -s file:///etc/sentry/sentry-site.xml -i file:///etc/sentry/sentry-provider.ini
 -u bob -e "select * from sample_08"
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
User bob has privileges to run the query
[root@server1 ~]# sentry --hive-config /etc/hive/conf --command config-tool
```

```
 -s file:///etc/sentry/sentry-site.xml -i file:///etc/sentry/sentry-provider.ini
 -u alice -e "select * from sample_08"
Using hive-conf-dir /etc/hive/conf
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:/etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
FAILED: SemanticException No valid privileges
*** Missing privileges for user alice:
         server=server1->db=default->table=sample_08->action=select
User alice does NOT have privileges to run the query
Sentry tool reported Errors: Compilation error: FAILED:
SemanticException No valid privileges
[root@server1 ~]#
```

7.7.5　从策略文件迁移

Sentry 服务被添加入项目时，还有一个有用的迁移工具也被同时加入。该工具使 6 管理员能够从已存在的文件向 Sentry 服务后端数据库导入策略。这缓解了需要为每一条规则导出 SQL 语法，并手动将它们添加进数据库的痛苦。该迁移工具是 `config-tool` 的功能增强，最后一节将进行讲解。示例 7-19 展示了该工具的用法。

示例 7-19　Sentry 策略导入工具

```
[root@server1 ~]# sentry --command config-tool --import \
-i file:///etc/sentry/sentry-provider.ini
Using hive-conf-dir /etc/hive/conf/
Configuration:
Sentry package jar: file:/var/lib/sentry/sentry-binding-hive-1.4.0.jar
Hive config: file:/etc/hive/conf/hive-site.xml
Sentry config: file:///etc/sentry/sentry-site.xml
Sentry Policy: file:///etc/sentry/sentry-provider.ini
Sentry server: server1
CREATE ROLE analyst_role;
GRANT ROLE analyst_role TO GROUP analysts;
# server=server1
GRANT SELECT ON DATABASE default TO ROLE analyst_role;
CREATE ROLE admin_role;
CREATE ROLE developer_role;
CREATE ROLE etl_role;
GRANT ROLE admin_role TO GROUP admins;
GRANT ALL ON SERVER server1 TO ROLE admin_role;
GRANT ROLE developer_role TO GROUP developers;
# server=server1
GRANT ALL ON DATABASE default TO ROLE developer_role;
GRANT ROLE etl_role TO GROUP etl;
# server=server1
GRANT INSERT ON DATABASE default TO ROLE etl_role;
# server=server1
GRANT SELECT ON DATABASE default TO ROLE etl_role;
[root@server1 ~]#
```

7.8 小结

从概念上讲，Sentry 常见易懂。然而从本章可以看出，知易行难。尽管 Sentry 是 Hadoop 生态系统中相对较新的组件之一，它依然很快成为 Hadoop 安全中不可分割的一部分。Sentry 在很短的一段时间内发生了迅速的演变，预计其他生态系统组件也会将 Sentry 集成，以便通过统一的方式提供强有力的授权控制。

我们已经顺利结束了关于验证和授权的庞大主题，下面看看审计，并理解用户活动在集群中的意义。

第8章

审计

目前为止，我们已经介绍了如何恰当地标识和验证用户和服务的身份，也阐述了怎样进行授权控制以限制用户和服务在集群中的行为。虽然这些各式各样的控制对定义和强化Hadoop 集群安全模型起到了很好的作用，但仍然缺乏安全模型的一个关键组件：审计。审计又称审核，是追踪集群中用户和服务行为的机制。这是安全问题中的一个关键部分。因为如果没有审计，那么任何人都可能察觉不到安全被破坏。审计功能提供对发生事情的记录以完善安全模型，可以被用在以下几个方面。

主动审计
: 这种审计与其他警报机制一起使用。例如，若一个用户试图访问集群中的资源并被拒，主动审计能够生成一封邮件发送给安全管理员，警告他们这件事情的发生。

被动审计
: 被动审计是指那些不会产生某些警报的审计。这通常是业务的最低要求，因此，指定的审计员和安全管理员可以通过查询审计寻找某些事件。例如，假如集群中存在一个安全漏洞，安全管理员能够通过查询审计日志找到漏洞被利用期间被访问的那些数据。

安全合规
: 一个业务可能需要被审计，从而使之满足内部合规性或法律合规性。这是最常见的情况：存储在 HDFS 中的数据包含敏感信息，比如个人身份信息（PII）、信用卡号和银行账号之类的财务信息、一些敏感的商业信息（工资单和工资财务）等。

Hadoop 的组件处理审计的方法各有不同，这取决于组件的功能。HDFS 和 HBase 这类组件是数据存储系统，因此它们的可审计事件集中于读、写和访问数据。与之相反，MapReduce、Hive 和 Impala 这类组件是查询引擎和处理框架，因此可审计事件集中于终端用户的查询和作业。后面几小节将深入介绍每个组件，并从审计的角度阐述与组件的典型交互。

8.1 HDFS审计日志

HDFS 提供两种不同的审计日志，以达到两种不同的目的。第一种日志是 hdfs-audit.log，用于对一般用户活动进行审计，例如用户创建新文件、改变文件权限、请求目录列表等。第二种日志是 SecurityAuth-hdfs.audit，用于审计服务层面的授权活动。这些日志文件的建立涉及对 log4j.category.SecurityLogger 以及 log4j.additivity.org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit 的钩取。示例 8-1 展示了此过程。

示例 8-1　HDFS log4j.properties

```
# other logging settings omitted
hdfs.audit.logger=${log.threshold},RFAAUDIT
hdfs.audit.log.maxfilesize=256MB
hdfs.audit.log.maxbackupindex=20
log4j.logger.org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit=
  ${hdfs.audit.logger}
log4j.additivity.org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit=false
log4j.appender.RFAAUDIT=org.apache.log4j.RollingFileAppender
log4j.appender.RFAAUDIT.File=${log.dir}/hdfs-audit.log
log4j.appender.RFAAUDIT.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAAUDIT.layout.ConversionPattern=%d{ISO8601} %p %c{2}: %m%n
log4j.appender.RFAAUDIT.MaxFileSize=${hdfs.audit.log.maxfilesize}
log4j.appender.RFAAUDIT.MaxBackupIndex=${hdfs.audit.log.maxbackupindex}
hadoop.security.logger=INFO,RFAS
hadoop.security.log.maxfilesize=256MB
hadoop.security.log.maxbackupindex=20
log4j.category.SecurityLogger=${hadoop.security.logger}
log4j.additivity.SecurityLogger=false
hadoop.security.log.file=SecurityAuth-${user.name}.audit
log4j.appender.RFAS=org.apache.log4j.RollingFileAppender
log4j.appender.RFAS.File=${log.dir}/${hadoop.security.log.file}
log4j.appender.RFAS.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAS.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
log4j.appender.RFAS.MaxFileSize=${hadoop.security.log.maxfilesize}
log4j.appender.RFAS.MaxBackupIndex=${hadoop.security.log.maxbackupindex}
```

那么，一个可审计事件出现时，到底发生了什么呢？对于这部分示例，先进行以下假设：

- 用户 Alice 被 Kerberos 规则 alice@EXAMPLE.COM 识别出身份，并且顺利地使用 kinit 获得一个有效的票据授予票据（TGT）；
- Alice 在 HDFS 的主目录中进行了显示目录列表的操作；
- Alice 在 HDFS 主目录中创建了一个名为 test 的空文件；
- Alice 将该文件的权限更改为全部可写；
- Alice 试图将文件从主目录移至 /user 路径。

示例 8-2 中，Alice 进行了一些在 HDFS 中的典型操作。这些都是用户活动事件，所以检查 hdfs-audit.log，以查看 Alice 通过 HDFS 操作留下的痕迹（示例中的 log 文件已经处理以适合阅读）。

示例 8-2　hdfs-audit.log

```
...
2014-03-11 23:50:18,251 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/alice dst=null perm=null
2014-03-11 23:50:18,280 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=listStatus src=/user/alice dst=null perm=null
2014-03-11 23:50:32,058 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/alice/test dst=null perm=null
2014-03-11 23:50:32,073 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/alice dst=null perm=null
2014-03-11 23:50:32,096 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=create src=/user/alice/test dst=null
 perm=alice:alice:rw-r-----
2014-03-11 23:50:39,558 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/alice/test dst=null perm=null
2014-03-11 23:50:39,587 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=setPermission src=/user/alice/test dst=null
 perm=alice:alice:rw-rw-rw-
2014-03-11 23:50:47,157 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user dst=null perm=null
2014-03-11 23:50:47,185 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/alice/test dst=null perm=null
2014-03-11 23:50:47,187 INFO FSNamesystem.audit: allowed=true ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=getfileinfo src=/user/test dst=null perm=null
2014-03-11 23:50:47,190 INFO FSNamesystem.audit: allowed=false ugi=alice@EXAMPLE.COM
 (auth:KERBEROS) ip=/10.1.1.1 cmd=rename src=/user/alice/test dst=/user/test perm=nul
...
```

如上所示，审计日志显示了 Alice 进行的每一项操作的相关信息。她进行的每一项操作首先都需要一个 `getfileinfo` 命令，接着是她所执行的多种操作内容（显示状态、创建、设置权限和重命名）。本日志中，产生事件的用户、事件发生的时间、执行操作的 IP 地址以及多种其他信息都很清楚。另外一个被记录的重要信息是，Alice 最近一次试图进行的操作是将文件从她的主目录移出至一个她无权访问的位置，且该操作被拒绝。

8.2　MapReduce审计日志

MapReduce 审计与 HDFS 审计很相似，包括两个目的相近的审计日志。第一个日志文件 `mapredaudit.log` 用于审计诸如作业提交之类的用户活动。第二个日志文件 `SecurityAuth-mapred.audit` 用于审计服务级授权活动，就像 HDFS 日志那样。这些文件的 `log4j` 参数需要进行设定，用于设定这些参数的钩子是 `log4j.category.SecurityLogger` 和 `log4j.logger.org.apache.hadoop.mapred.AuditLogger`，如示例 8-3 所示。

示例 8-3　MapReduce log4j.properties

```
# other logging settings omitted
hadoop.security.logger=INFO,RFAS
hadoop.security.log.maxfilesize=256MB
hadoop.security.log.maxbackupindex=20
log4j.category.SecurityLogger=${hadoop.security.logger}
log4j.additivity.SecurityLogger=false
hadoop.security.log.file=SecurityAuth-${user.name}.audit
```

```
log4j.appender.RFAS=org.apache.log4j.RollingFileAppender
log4j.appender.RFAS.File=${log.dir}/${hadoop.security.log.file}
log4j.appender.RFAS.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAS.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
log4j.appender.RFAS.MaxFileSize=${hadoop.security.log.maxfilesize}
log4j.appender.RFAS.MaxBackupIndex=${hadoop.security.log.maxbackupindex}
mapred.audit.logger=${log.threshold},RFAAUDIT
mapred.audit.log.maxfilesize=256MB
mapred.audit.log.maxbackupindex=20
log4j.logger.org.apache.hadoop.mapred.AuditLogger=${mapred.audit.logger}
log4j.additivity.org.apache.hadoop.mapred.AuditLogger=false
log4j.appender.RFAAUDIT=org.apache.log4j.RollingFileAppender
log4j.appender.RFAAUDIT.File=${log.dir}/mapred-audit.log
log4j.appender.RFAAUDIT.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAAUDIT.layout.ConversionPattern=%d{ISO8601} %p %c{2}: %m%n
log4j.appender.RFAAUDIT.MaxFileSize=${mapred.audit.log.maxfilesize}
log4j.appender.RFAAUDIT.MaxBackupIndex=${mapred.audit.log.maxbackupindex}
```

对于这个示例，进行以下假设：

- 用户 Bob 被 Kerberos 规则 `bob@EXAMPLE.COM` 识别出身份，并且已经成功地使用 `kinit` 获得一个有效的 TGT；
- 没有使用 MapReduce 服务级授权；
- Bob 提交了一个 MapReduce 作业；
- Bob 在 MapReduce 作业结束以前终止了它。

日志中这一系列操作的结果如示例 8-4 和示例 8-5 所示。

示例 8-4　mapred-audit.log

```
...
2014-03-12 18:11:46,363 INFO mapred.AuditLogger: USER=bob IP=10.1.1.1
OPERATION=SUBMIT_JOB TARGET=job_201403112320_0001 RESULT=SUCCESS
...
```

示例 8-5　SecurityAuth-mapred.audit

```
...
2014-03-12 18:46:25,200 INFO SecurityLogger.org.apache.hadoop.ipc.Server:
 Auth successful for bob@EXAMPLE.COM (auth:SIMPLE)

2014-03-12 18:46:25,239 INFO SecurityLogger.org.apache.hadoop.security.
 authorize.ServiceAuthorizationManager: Authorization successful for
 bob@EXAMPLE.COM (auth:KERBEROS) for protocol=interface
org.apache.hadoop.mapred.JobSubmissionProtocol

2014-03-12 18:46:29,955 INFO SecurityLogger.org.apache.hadoop.ipc.Server:
 Auth successful for job_201403112320_0002 (auth:SIMPLE)

2014-03-12 18:46:29,976 INFO SecurityLogger.org.apache.hadoop.security.
 authorize.ServiceAuthorizationManager: Authorization successful for
 job_201403112320_0002 (auth:TOKEN) for protocol=interface
 org.apache.hadoop.mapred.TaskUmbilicalProtocol

...(more)...
```

```
2014-03-12 18:47:11,598 INFO SecurityLogger.org.apache.hadoop.ipc.Server:
 Auth successful for bob@EXAMPLE.COM (auth:SIMPLE)

2014-03-12 18:47:11,638 INFO SecurityLogger.org.apache.hadoop.security.
 authorize.ServiceAuthorizationManager: Authorization successful for
 bob@EXAMPLE.COM (auth:KERBEROS) for protocol=interface
 org.apache.hadoop.mapred.JobSubmissionProtocol
...
```

示例 8-4 很容易看懂：用户 Bob 执行了操作 `SUBMIT_JOB`，这导致 ID 为 `job_201403112320_0001` 的 MapReduce 作业的创建。正如预期的那样，其他相关信息包括事件发生的时间日期以及 IP 地址。示例 8-5 中，情况有所不同。第一个条目显示 Bob 成功通过 JobTracker 的验证，第二个条目表明 Bob 已经被授权提交该作业。接下来的两个事件（为了简洁起见，其后的两个相同事件已被删去）显示了作业本身的操作。有趣的是，Bob 终止正在运行的作业时，出现的审计事件与其提交该作业时的审计事件一模一样。

8.3 YARN审计日志

YARN 审计日志事件穿插在守护进程的日志文件中，然而它们更容易辨识，因为它们的类名称会被事件记录。对于 Resource Manager，它的名称是 `org.apache.hadoop.yarn.server.resourcemanager.RMAuditLogger`；对于 Node Manager，则是 `org.apache.hadoop.yarn.server.nodemanager.NMAuditLogger`。可以使用这些类名从普通的应用程序日志中解析审计事件。YARN 若要记录审计日志，需要对 `log4j` 参数进行设置。该操作的钩子是 `log4j.category.SecurityLogger`，示例 8-6 示范了如何进行设置。

示例 8-6　YARN log4j.properties

```
# other logging settings omitted
hadoop.security.logger=INFO,RFAS
hadoop.security.log.maxfilesize=256MB
hadoop.security.log.maxbackupindex=20
log4j.category.SecurityLogger=${hadoop.security.logger}
log4j.additivity.SecurityLogger=false
hadoop.security.log.file=SecurityAuth-${user.name}.audit
log4j.appender.RFAS=org.apache.log4j.RollingFileAppender
log4j.appender.RFAS.File=${log.dir}/${hadoop.security.log.file}
log4j.appender.RFAS.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAS.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
log4j.appender.RFAS.MaxFileSize=${hadoop.security.log.maxfilesize}
log4j.appender.RFAS.MaxBackupIndex=${hadoop.security.log.maxbackupindex}
```

本例中，用户 Alice 通过 YARN 提交了一个 MapReduce 作业，紧接着作业开始运行。示例 8-7 展示了针对 Resource Manager 的审计事件，示例 8-8 展示了 Node Manager 之一的审计事件。注意，这里为了简洁起见而省略了重复的审计类名，并且事件日志已经以适合阅读。

示例 8-7　YARN Resource Manager 审计事件

```
2014-12-27 12:49:35,182 INFO USER=alice IP=10.6.9.73
  OPERATION=Submit Application Request
```

```
  TARGET=ClientRMService RESULT=SUCCESS
  APPID=application_1419453547005_0001
2014-12-27 12:49:43,598 INFO USER=alice OPERATION=AM Allocated Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000001
2014-12-27 12:49:57,288 INFO USER=alice IP=10.6.9.75 OPERATION=Register App Master
  TARGET=ApplicationMasterService RESULT=SUCCESS APPID=application_1419453547005_0001
  APPATTEMPTID=appattempt_1419453547005_0001_000001
2014-12-27 12:50:02,375 INFO USER=alice OPERATION=AM Allocated Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000002
2014-12-27 12:50:02,376 INFO USER=alice OPERATION=AM Allocated Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000003
2014-12-27 12:50:19,361 INFO USER=alice OPERATION=AM Released Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000002
2014-12-27 12:50:21,436 INFO USER=alice OPERATION=AM Released Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000003
2014-12-27 12:50:27,954 INFO USER=alice OPERATION=AM Released Container
  TARGET=SchedulerApp RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000001
2014-12-27 12:50:27,963 INFO USER=alice OPERATION=Application Finished - Succeeded
  TARGET=RMAppManager RESULT=SUCCESS APPID=application_1419453547005_0001
```

示例 8-8　YARN Node Manager 审计事件

```
2014-12-27 12:49:43,956 INFO USER=alice IP=10.6.9.75
  OPERATION=Start Container Request TARGET=ContainerManageImpl RESULT=SUCCESS
  APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000001
2014-12-27 12:50:27,105 INFO USER=alice OPERATION=Container Finished - Succeeded
  TARGET=ContainerImpl RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000001
2014-12-27 12:50:27,984 INFO USER=alice IP=10.6.9.75 OPERATION=Stop Container Request
  TARGET=ContainerManageImpl RESULT=SUCCESS APPID=application_1419453547005_0001
  CONTAINERID=container_1419453547005_0001_01_000001
```

YARN 的众多优点之一是，能够指定资源池。如前所述，资源池可以进行授权控制，从而使得只有特定的用户和组才能够提交给特定的资源池。接下来的示例中，Bob 试图向 Prod 资源池提交数据，但他没有权限。示例 8-9 展示了这种情况下的审计日志。再次强调，为了简洁起见而省略了重复的审计类名，并且日志已经处理以适合阅读。

示例 8-9　YARN Resource Manager 审计事件

```
2014-12-27 13:56:35,886 INFO USER=bob IP=10.6.9.73
  OPERATION=Submit Application Request TARGET=ClientRMService
  RESULT=SUCCESS APPID=application_1419705820412_0002
2014-12-27 13:56:35,917 WARN USER=bob OPERATION=Application Finished - Failed
  TARGET=RMAppManager RESULT=FAILURE DESCRIPTION=App failed with state: FAILED
  PERMISSIONS=User bob cannot submit applications to queue root.prod
  APPID=application_1419705820412_0002
```

8.4 Hive审计日志

Hive 审计与 YARN 相似，没有专用的审计日志文件。审计事件实际发生在 Hive Metastore 服务日志内部，因此安全管理员从常规应用的日志事件获取相关审计信息会有些困难。然而同 YARN 一样，可以使用审计记录的类名辨识审计事件。其他 Hive 组件——如 HiveServer2——没有专用的审计，但仍然能从服务日志找到类似审计的信息。对于该例假设：

- 用户 Bob 被 Kerberos 规则 `bob@EXAMPLE.COM` 识别出身份，并且已经成功地使用 kinit 获得一个有效的 TGT；
- Bob 使用 `beeline` CLI 连接至 HiveServer2；
- Bob 首先执行 `show tables` 命令，以列出默认数据库中的表；
- 然后 Bob 执行 `select count(*) from sample_07`，以统计 `sample_07` 表中的记录数。

这些操作的结果如示例 8-10 所示，该例已经调整了可读性。

示例 8-10　Hive Metastore 审计事件

```
...
2014-03-29 17:13:18,778 INFO org.apache.hadoop.hive.metastore.HiveMetaStore.audit:
 ugi=bob        ip=/10.1.1.1    cmd=get_database: default
...
2014-03-29 17:13:18,782 INFO org.apache.hadoop.hive.metastore.HiveMetaStore.audit:
 ugi=bob        ip=/10.1.1.1    cmd=get_tables: db=default pat=.*
...
2014-03-29 17:13:37,110 INFO org.apache.hadoop.hive.metastore.HiveMetaStore.audit:
 ugi=bob        ip=/10.1.1.1    cmd=get_table : db=default tbl=sample_07
```

看看示例 8-10 中的审计事件显示的一些内容。首先，审计事件本身由 `org.apache.hadoop.hive.metastore.HiveMetaStore.audit` 标记，这使从日志搜索审计事件变得容易了一些。其次，这些审计事件在用户身份识别方面与之前见过的审计事件有着细微的区别。Hive 仅显示用户名，而不是 Kerberos UPN（User Principal Name）全名。每个审计事件中，用户执行的操作是由 `cmd` 字段标识的。可以看到，`show tables` 的查询生成两个审计事件：`get_database` 和 `get_tables`。用于统计行数的实际 SQL 查询生成了一个审计事件，即 `get_table`。与其他组件中见过的审计事件一样，本审计事件也给出了执行操作的用户 IP 地址。

8.5 Cloudera Impala审计日志

Impala 审计事件被 Impala 守护进程（impalad）记录到专用的审计日志。审计日志目录位置由标识 `audit_event_log_dir` 指定，一个典型的选择是 `/var/log/impalad/audits`。这些日志文件到达由指定行数决定的一定大小后，将会滚动，文件大小由标识 `max_audit_event_log_file_size` 设定，通常合理的设定是 5000 行。对于 Impala 的例子，做出与 Hive 的例子一模一样的场景假设。这些操作的结果如示例 8-11 所示。

示例 8-11　Impala 守护进程的审计日志

```
....

{"1396114935263":{"query_id":"914b9eb1591546f0:ff4419eab4de439c",
 "session_id":"e643b5e102f653ec:94e0a3d4b3646ca3",
 "start_time":"2014-03-29 17:42:15.201945000","authorization_failure":false,
 "status":"","user":"bob","impersonator":null,"statement_type":"SHOW_TABLES",
 "network_address":"::ffff:10.1.1.1:47569","sql_statement":"show tables",
 "catalog_objects":[]}}

{"1396115148996":{"query_id":"97443eddd3c172fd:34fe3f37c84d6ea8",
 "session_id":"e643b5e102f653ec:94e0a3d4b3646ca3",
 "start_time":"2014-03-29 17:45:48.850540000","authorization_failure":false,
 "status":"","user":"bob","impersonator":null,"statement_type":"QUERY",
 "network_address":"::ffff:10.1.1.1:47569","sql_statement":
 "select count(*) from sample_07","catalog_objects":
[{"name":"default.sample_07","object_type":"TABLE","privilege":"SELECT"}]}}
....
```

示例 8-11 显示出，Impala 的审计日志与其他 Hadoop 组件的日志在格式上截然不同。这些审计事件以 JSON 格式记录，因此可读性较差，但借助外部工具可以轻松使用这些数据。第一个审计事件展示了用户在 `statement_type` 字段中设定的操作类型，即 `SHOW_TABLES`。该信息也能用于 `sql_statement` 字段，以展示 Bob 进行的精确查询操作。第二个审计事件展示了*查询*的操作类型。

8.6　HBase审计日志

HBase 日志将审计事件记录到一个单独的日志文件，可以在相关的 `log4j.properties` 文件中对该日志文件进行配置。HBase 的架构是：客户端仅与负责执行指定操作的特定服务器进行交互，因此审计事件在整个 HBase 集群中都有分布。例如，创建、删除和修改表的操作是由 HBase Master 负责的；而类似扫描、插入和读取这种方法是指定给表中的某个特定区域的，所以由 RegionServer 记录这些事件。

HBase 对审计事件进行记录时，需要对 `log4j` 参数进行设置。用于进行该设置的钩子为 `log4j.logger.SecurityLogger`，示例 8-12 展示了如何进行设置。

示例 8-12　HBase log4j.properties

```
# other logging settings omitted
log4j.logger.SecurityLogger=TRACE, RFAS
log4j.additivity.SecurityLogger=false
log4j.appender.RFAS=org.apache.log4j.RollingFileAppender
log4j.appender.RFAS.File=${log.dir}/audit/SecurityAuth-hbase.audit
log4j.appender.RFAS.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAS.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
log4j.appender.RFAS.MaxFileSize=${max.log.file.size}
log4j.appender.RFAS.MaxBackupIndex=${max.log.file.backup.index}
```

本例进行了以下操作：

- HBase 超级用户创建了一个名为 sample 的表；
- HBase 超级用户授予用户 Alice 对 sample 表的 RW（读写）访问权限；
- HBase 超级用户授予用户 Bob 对 sample 表的 R（读）访问权限；
- Alice 试图创建一个名为 sample2 的新表，但被拒绝访问；
- Alice 向 sample 表插入一个值；
- Alice 扫描 sample 表；
- Bob 扫描 sample 表；
- Bob 试图向 sample 表插入一个值，但被拒绝访问。

HBase 审计能够缩小到一个特定的类，即 SecurityLogger.org.apache.hadoop.hbase.security.access.AccessController，和其他组件中的日志事件一样。这个类贯穿于各类日志，但为了简洁起见，将它从示例 8-13 和 8-14 中略去。同样，为了保证可读性，这些示例的格式都经过了编辑。

示例 8-13　HBase master 的审计日志

```
2014-12-27 21:05:56,938 TRACE Access allowed for user hbase; reason:
  Global check allowed; remote address: /10.6.9.74; request: createTable;
  context: (user=hbase@EXAMPLE.COM, scope=sample, family=cf, action=CREATE)
2014-12-27 21:06:09,484 TRACE Access allowed for user hbase; reason:
  Table permission granted; remote address: /10.6.9.74;
  request: getTableDescriptors; context: (user=hbase@EXAMPLE.COM,
  scope=sample, family=, action=ADMIN)
2014-12-27 21:06:16,620 TRACE Access allowed for user hbase; reason:
 Table permission granted; remote address: /10.6.9.74;
 request: getTableDescriptors; context: (user=hbase@EXAMPLE.COM,
 scope=sample, family=, action=ADMIN)
2014-12-27 21:07:02,102 TRACE Access denied for user alice; reason:
  Global check failed; remote address: /10.6.9.74; request:
  createTable; context: (user=alice@EXAMPLE.COM, scope=sample2,
  family=cf, action=CREATE)
```

由示例 8-13 可知，表创建的时间被清楚地记录下来。用户 hbase 被允许——而 Alice 被拒绝——创建一个表。此外，还可以从这个日志文件中看出，实际授予权限的行为并不明显。当该操作被记录为一个 ADMIN 操作，并且与这个表有同样的作用域，那么会没有任何标识表明该表权限被授予哪个用户。这是 HBase 的一个局限，很可能在将来的版本中得到改善。

示例 8-14　HBase region server 审计日志

```
2014-12-27 21:07:15,411 TRACE Access allowed for user alice; reason:
  Table permission granted; remote address: /10.6.9.74; request: put;
  context: (user=alice@EXAMPLE.COM, scope=sample,
  family=cf:col1, action=WRITE)
2014-12-27 21:07:18,705 TRACE Access allowed for user alice; reason:
  Table permission granted; remote address: /10.6.9.74; request: scan;
  context: (user=alice@EXAMPLE.COM, scope=sample, family=cf, action=READ)
2014-12-27 21:07:47,263 TRACE Access allowed for user bob; reason:
  Table permission granted; remote address: /10.6.9.74; request: scan;
  context: (user=bob@EXAMPLE.COM, scope=sample, family=cf, action=READ)
```

```
2014-12-27 21:07:57,756 TRACE Access denied for user bob; reason:
  Failed qualifier check; remote address: /10.6.9.74; request: put;
  context: (user=bob@EXAMPLE.COM, scope=sample, family=cf:col1, action=WRITE)
```

示例 8-14 中，Alice 和 Bob 尝试进行的读和写操作被清楚地标识出来。它提供了表的相关信息、列族、列以及这项操作被允许或拒绝的原因。

8.7　Accumulo审计日志

与 HBase 相似，Accumulo 可以被配置为将审计事件记录到一个单独的日志文件。由于不要求 Accumulo 客户端在每次访问时都与一个单一、集中式的服务通信，因此审计日志分散于整个集群。例如，用户创建、删除或修改表时，操作会被 Accumulo Master 记录，而扫描和写入这类操作则会被实际处理操作的 TabletServer 记录。

默认的 Accumulo 配置模板是将审计记录关闭的。用户可以在 log4j 配置文件 `auditLog.xml` 中，将审计记录的日志级别设为 `INFO`，以启用记录功能。示例 8-15 展示了一个启用审计记录功能的示例文件 auditLog.xml。

示例 8-15　Accumulo auditLog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <!-- Write out Audit info to an Audit file -->
  <appender name="Audit" class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File"
     value="/var/log/accumulo/accumulo01.example.com.audit"/>
    <param name="MaxBackupIndex" value="10"/>
    <param name="DatePattern" value="'.'yyyy-MM-dd"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
       value="%d{yyyy-MM-dd HH:mm:ss,SSS/Z} [%c{2}] %-5p: %m%n"/>
    </layout>
  </appender>
  <logger name="Audit" additivity="false">
    <appender-ref ref="Audit" />
    <!-- Change level from OFF to INFO. default:<level value="OFF"/> -->
    <level value="INFO"/>
  </logger>

</log4j:configuration>
```

Accumulo 既审计普通用户的访问，也审计系统管理员的操作。每条审计记录包括操作状态（成功、失败、允许、拒绝）和进行该操作的用户，如果操作是远程请求，则还包括客户端地址。对于失败的请求，还会记录那些导致失败的异常。虽然个体的操作行为在细节上有所不同，但通常都会包含操作目标，以及诸如被访问的行和列范围之类的相关参数这种细节。表 8-1 展示了 Accumulo 记录的行为列表。

表8-1：Accumulo的审计操作

操作	描述
authenticate	用户通过 Accumulo 的身份验证
createUser	管理员创建一个新用户
dropUser	管理员删除用户
changePassword	管理员更换用户的密码
changeAuthorizations	管理员改变对用户的授权
grantSystemPermission	管理员将系统权限授予用户
grantTablePermission	管理员授予用户某表的权限
revokeSystemPermission	管理员撤销用户的系统授权
revokeTablePermission	管理员撤销用户对某表的权限
createTable	用户创建表
deleteTable	用户删除表
renameTable	用户对表进行重命名
cloneTable	用户复制表
scan	用户扫描一定范围的行内容
deleteData	用户删除表数据
bulkImport	用户发起批量数据导入
export	用户从一个集群向另一集群导出表
import	用户导入一张从别处导出的表

现在看看进行一些操作之后的审计日志。本例包含执行以下操作之后的审计结果：

- Accumulo 的 root 用户创建了名为 `alice` 的用户；
- Accumulo 的 root 用户创建了名为 `bob` 的用户；
- Accumulo 的 root 用户创建了名为 `sample` 的表；
- Accumulo 的 root 用户授予 `alice` 对 `sample` 表的 `Table.READ` 和 `Table.WRITE` 访问权限；
- Accumulo 的 root 用户授予 `bob` 对 `sample` 表的 `Table.READ` 访问权限；
- alice 试图创建新表 `sample2`，访问被拒绝；
- alice 向 `sample` 表插入一个值；
- alice 扫描了 `sample` 表；
- bob 扫描了 `sample` 表；
- bob 试图向 `sample` 表插入一个值，访问被拒绝。

示例 8-16 和示例 8-17 中的审计日志根据可读性进行了格式上的调整，但其他内容没有改动。

示例 8-16　Accumulo master 的审计日志

```
2014-12-27 16:40:11,673/-0800 [Audit] INFO : operation: permitted;
    user: root; action: createTable; targetTable: sample;
2014-12-27 16:40:28,563/-0800 [Audit] INFO : operation: denied;
    user: alice; action: createTable; targetTable: sample2;
```

示例 8-16 中可以看到，创建表的操作被清晰地记录下来。`root` 用户能被允许进行

createTable 操作，然而 alice 被拒绝。其他的管理员操作被记录在 TabletServer 日志中。

示例 8-17　Accumulo TabletServer 的审计日志

```
2014-12-27 16:39:49,262/-0800 [Audit] INFO : operation: success;
    user: root: action: createUser; targetUser: alice; Authorizations: ;
2014-12-27 16:40:02,226/-0800 [Audit] INFO : operation: success;
    user: root: action: createUser; targetUser: bob; Authorizations: ;
2014-12-27 16:40:13,226/-0800 [Audit] INFO : operation: success;
    user: root: action: grantTablePermission; permission: READ;
    targetTable: sample; targetUser: alice;
2014-12-27 16:40:13,292/-0800 [Audit] INFO : operation: success;
    user: root: action: grantTablePermission; permission: WRITE;
    targetTable: sample; targetUser: alice;
2014-12-27 16:40:13,442/-0800 [Audit] INFO : operation: success;
    user: root: action: grantTablePermission; permission: READ;
    targetTable: sample; targetUser: bob;
2014-12-27 16:40:30,529/-0800 [Audit] INFO : operation: permitted;
    user: alice; action: scan; targetTable: sample; authorizations: ;
    range: (-inf,+inf); columns: []; iterators: []; iteratorOptions: {};
2014-12-27 16:40:43,180/-0800 [Audit] INFO : operation: permitted;
    user: bob; action: scan; targetTable: sample; authorizations: ;
    range: (-inf,+inf); columns: []; iterators: []; iteratorOptions: {};
```

示例 8-17 展示了 TabletServer 审计日志的输出能看到 createUser 操作、被创建的用户和该用户被授予的权限，也可以看到经过授权的 grantTablePermission 操作、目标表和目标用户。最后还可以看到两个 scan 操作和查询的细节：行范围、列和所用迭代器。值得注意的是写操作的缺失，这是 Accumulo 审计框架当前的不足之处。同样，也看不到身份验证事件，因为这些事件是被 shell 自身记录的。

8.8　Sentry审计日志

第 7 章中，我们知道 Sentry 的最新版本使用一个服务处理授权请求，并管理与规则库的交互。审计过程中，对修改授权策略的事件进行审计是极其关键的。为了做到这一点，Sentry 需要被配置为捕获审计事件，而记录类 sentry.hive.authorization.ddl.logger 正是需要被配置的类之一。示例 8-18 展示了如何进行这种配置。

示例 8-18　Sentry 服务的 og4j.properties

```
# other log settings omitted
log4j.logger.sentry.hive.authorization.ddl.logger=${sentry.audit.logger}
log4j.additivity.sentry.hive.authorization.ddl.logger=false
sentry.audit.logger=TRACE,RFAAUDIT
sentry.audit.log.maxfilesize=256MB
sentry.audit.log.maxbackupindex=20
log4j.appender.RFAAUDIT=org.apache.log4j.RollingFileAppender
log4j.appender.RFAAUDIT.File=${log.dir}/audit/sentry-audit.log
log4j.appender.RFAAUDIT.layout=org.apache.log4j.PatternLayout
log4j.appender.RFAAUDIT.layout.ConversionPattern=%d{ISO8601} %p %c{2}: %m%n
log4j.appender.RFAAUDIT.MaxFileSize=${sentry.audit.log.maxfilesize}
log4j.appender.RFAAUDIT.MaxBackupIndex=${sentry.audit.log.maxbackupindex}
```

现在，Sentry 被配置为记录审计事件，下面看一个示例。该例中，Alice 是 Sentry 管理员，Bob 不是。Alice 使用 `beeline` 控制台创建了一个名为 `analyst` 的新角色，并将它分配给 `analystgrp` 组，然后授予它对默认数据库进行 `SELECT` 操作的特权。接下来，Bob 试图使用 `impala-shell` 创建一个新角色，但访问被拒绝。示例 8-19 展示了这些行为的记录。

示例 8-19　Sentry 服务审计日志

```
2015-01-02 11:17:10,753 INFO ddl.logger:
  {"serviceName":"Sentry-Service","userName":"alice","impersonator":
  "hive/server1.example.com@EXAMPLE.COM","ipAddress":"/10.6.9.74",
  "operation":"CREATE_ROLE","eventTime":"1420215430742","operationText":
  "CREATE ROLE analyst","allowed":"true","databaseName":null,
  "tableName":null,"resourcePath":null,"objectType":"ROLE"}
2015-01-02 11:17:37,537 INFO ddl.logger:
  {"serviceName":"Sentry-Service","userName":"alice","impersonator":
  "hive/server1.example.com@EXAMPLE.COM","ipAddress":"/10.6.9.74",
  "operation":"ADD_ROLE_TO_GROUP","eventTime":"1420215457536",
  "operationText":"GRANT ROLE analyst TO GROUP analystgrp","allowed":"true",
  "databaseName":null,"tableName":null,"resourcePath":null,"objectType":"ROLE"}
2015-01-02 11:17:52,408 INFO ddl.logger:
  {"serviceName":"Sentry-Service","userName":"alice","impersonator":
  "hive/server1.example.com@EXAMPLE.COM","ipAddress":"/10.6.9.74",
  "operation":"GRANT_PRIVILEGE","eventTime":"1420215472407","operationText":
  "GRANT SELECT ON DATABASE default TO ROLE analyst","allowed":"true",
  "databaseName":"default","tableName":"","resourcePath":"","objectType":"PRINCIPAL"}
2015-01-02 11:33:20,199 INFO ddl.logger:
  {"serviceName":"Sentry-Service","userName":"bob","impersonator":
  "impala/server1.example.com@EXAMPLE.COM","ipAddress":"/10.6.9.73",
  "operation":"CREATE_ROLE","eventTime":"1420216400199","operationText":
  "CREATE ROLE temp","allowed":"false","databaseName":null,"tableName":null,
  "resourcePath":null,"objectType":"ROLE"}
```

如日志所示，实际的审计记录是 JSON 格式的。该格式有利于外部的日志聚合和管理系统对日志进行处理，这在大型企业中是很重要的。

8.9　日志聚合

审计日志通常分散在集群中的许多节点（如果不是全部）。节点的数量乘以生成的独立日志文件的数量，使得掌握系统中正在发生的事情变成一项庞大的任务。强烈推荐的典型做法是，使用某种日志聚合系统从集群中的所有节点拉取所有审计事件，放入一个集中化的位置进行存储和分析。当然，还有专门面向 Hadoop 的方法，以获取关于集群中发生事件的额外信息。即便如此，企业中已经部署的通用日志聚合系统也是管理 Hadoop 审计日志的好办法。

另一个有意思的日志聚合选择是，将其聚合后再导回 Hadoop 集群以进行分析。Hadoop 的安全用例是通用的，在 Hadoop 中分析审计事件也适合这些用例。如本章所示，审计事件通常都是结构化形式，以方便使用类似 Hive 或 Impala 的 SQL 工具进行查询。

8.10　小结

本章学习了 Hadoop 生态系统中的几种组件，并且阐述了用户与集群交互时被记录审计事件的类型。这些日志事件不仅对于审计普通用户在做什么很关键，对于发现非授权用户试图做什么也很关键。尽管 Hadoop 生态系统没有原生警告功能，但日志事件的架构有利于外部附加工具以一种更通用化的方式使用事件日志。主动警告是 Hadoop 生态系统正在开发的新功能，尽管如此，仍然有很多通用的日志聚合工具拥有对满足某种条件的事件进行警告的功能，这些工具在企业中很通用。

本章涉及的所有审计功能包含 AAA（认证、授权和审计）中的审计部分。之后将深入学习实际的数据——Hadoop 的生命线和大数据——是怎样受保护的。

第三部分

数据安全



第 9 章
数据保护

Eddie Garcia

目前为止，我们已经讲述了如何配置 Hadoop 以实施标准的 AAA 控制。本章将理解这些控制如何与第 1 章介绍的 CIA 准则一起，为数据保护提供基础。数据保护是一个宽泛的概念，包括从数据保密性到准用性等很多主题。其中需要特别关注的一个内容是*加密*。

加密是保护数据的一种常见方法。数据加密主要有两种类型：*静态数据*（**data-at-rest**）*加密*和*动态数据*（**data-in-transit**）*加密*，也称为*线上*（**over-the-wire**）*加密*。静态数据指的是，即使机器关闭后依旧可以存储的数据，包括硬盘、闪存盘、USB 记忆棒、内存卡、CD、DVD，甚至储藏箱中一些老式软盘或磁带中的数据。动态数据，顾名思义，是流动的数据，例如在互联网、USB 线、咖啡店 Wi-Fi、手机基站或者从远程空间站到地球之间传输的数据。

9.1 加密算法

深入了解两种类型的数据加密之前，先简要介绍加密算法。加密算法定义了用于加密数据的数学方法。一种常见的加密算法是 **AES**（**Advanced Encryption Standard**，*高级加密标准*），它是由美国国家标准技术研究所（NIST）在 FIPS-197（http://1.usa.gov/1GFoldU）中建立的一个标准。

本书不会描述 AES 加密是如何工作的，推荐阅读 Christof Paar 和 Jan Palzl 编著的《深入浅出密码学：常用加密技术原理与应用》中的第 4 章。其他常见的加密算法还包括 DES、RC4、Twofish 和 Blowfish。

加密数据时，密钥的大小很重要。通常来说，密钥越大，越难破解，但缺点是加密速度就越慢。处理很小的数据时，加密密钥的大小应该不超过数据本身。

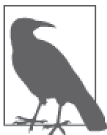
使用 AES 时，所支持的常见密钥大小是 128 位、192 位和 256 位。当今的业界标准是 AES-256（256 位密钥）加密，但历史已经证明，这也是会变的。DES 和三重 DES（3 轮 DES 加密）一度是业界标准，但这两种加密算法都能被现在的计算机很轻易地暴力破解。

考虑到加密带来的性能开销，芯片制造商创造了硬件层函数以提升加密性能。这些增强可以使加密速度比软件加密提升多个数量级。一种流行的硬件加密技术是英特尔的 AES-NI。

对加密方法和算法有了基本了解之后，可以深入探讨全盘加密和文件系统加密。这些加密方法不需要特殊硬件，实现起来较为容易。

9.2 静态数据加密

假设 Alice 将一条给 Bob 的消息放在一个 USB 记忆棒中，并将其给了 Bob。但 Bob 在慌乱中不知道将这个 USB 记忆棒放到哪儿了，而 Eve 恰好捡到了它。出于好奇，Eve 将其连到 USB 上，尝试读取其中内容。幸运的是，Alice 对消息进行了加密，否则她就会陷入尴尬的局面。这个简单的例子中，加密保证了消息的机密性，除 Bob 之外，没有人知道解密数据的密码。Hadoop 生态系统的核心是 HDFS，它是很多其他组件的文件系统。直到最近，HDFS 本身才支持加密数据，这意味着需要采用其他加密方法。



这些年来，经常发生由笔记本电脑和手机丢失、硬盘处理不当、硬件被盗导致的敏感数据泄漏案件。静态数据加密能够减轻这些类型的数据泄漏，因为加密使得查看数据变得更加困难（但并非不可能）。

除了原生 HDFS 加密之外，下面看看另外三种选择。我们不会对每个方法深入探讨，因为一些方法是针对厂商的。这些方法在 HDFS 之下以透明的方式工作，因此不需要进行 Hadoop 相关配置。所有这些方法都会在某个磁盘从磁盘阵列中物理移除的情况下对数据进行保护。

加密磁盘

该方法与操作系统完全无关，HDFS 用于存储数据的物理磁盘，支持原生加密。不足之处在于，加密磁盘无法保护数据不被恶意用户和系统上运行的进程读取。

全盘加密

该方法通常在系统启动时工作，与加密磁盘方式不同的是，它不需要特殊的磁盘或硬件。这种技术存在若干种实现方案，通常根据操作系统的不同而不同。一些全盘加密方法支持操作系统根分区加密，一些只支持 HDFS 块所在的数据分区或数据卷加密。全盘加密也与加密磁盘方式有一样的不足，即无法保护数据不被恶意用户和系统上运行的进程读取。

文件系统加密

该方法在操作系统层工作，不需要特殊的磁盘或硬件。这种技术存在若干种实现方案，根据操作系统的不同而不同。该方法不支持对根分区的文件系统加密，否则就变成“鸡生蛋还是蛋生鸡”的问题了：加密 OS 需要先启动，以对 OS 进行解密。文件系统加密的一个好处是，它能够保护数据不被恶意用户和系统上运行的进程窃取。如果一个加密

的主目录使用一个某用户知道的密码进行加密，并且该用户自系统启动之后没有登录过，那么恶意用户或进程就不可能获得该密码解锁用户数据，即便是 root 用户也不行。

9.2.1 加密和密钥管理

Hadoop 生产集群通常有成千上万个节点，每个节点有 8~12 个磁盘，将静态数据加密扩展到 HDFS 之外的其他组件增大了潜在的复杂性。准备实施加密时，考虑如下几个额外的问题至关重要：

- 如何在加密情况下配置多个磁盘分区？
- 如何避免在系统启动时或在明文脚本中提供密码？

最后，大规模静态加密的难点在于密钥管理。下一节将要讨论的原生 HDFS 静态数据加密混合使用了两种方式：将加密密钥与文件元数据组合，以及依赖外部的 key server 管理密钥材料。我们讨论的其他静态数据加密技术一定程度上也需要密钥管理服务。

为密钥管理系统选择厂商很复杂，我们也无法为你的环境推荐某个厂商。但你选择时可以考虑如下几个重要标准：

- 解决方案是否支持硬件安全模块？
- 解决方案的扩展性如何（密钥个数以及每秒获取密钥的速度）？
- 解决方案是否支持 Hadoop 标准（如 KeyProvider 接口）？
- 管理成百上千个密钥的授权控制的难易程度如何？

如果正在使用预打包的 Hadoop 版本，那么寻找密钥管理系统厂商最简单的方法是，查找你的 Hadoop 厂商所认证的安全厂商。

9.2.2 HDFS静态数据加密

从 Hadoop 2.6 开始，HDFS 支持原生静态加密。这项功能并不被视为全盘加密或文件系统加密，而是另外一种名为*应用层加密*的类型。该方法中，数据被发送和到达存储位置之前，在应用层被加密。这种方法运行在操作系统层之上，不需要除 Hadoop 提供之外的任何特殊的操作系统软件包或硬件。要想了解更多关于原生 HDFS 加密设计的信息，可以阅读 HDFS 静态数据加密设计文档（http://bit.ly/1KZMeph）。

HDFS 中，需要加密的目录被分解成若干*加密区*（**encryption zone**），加密区中的每个文件都被使用唯一的*数据加密密钥*（**data encryption key**，**DEK**）进行加密，这就是加密区的区别所在。明文 DEK *不是*永久的，而是用一个名为*加密区密钥*（**encryption zone key**，**EZK**）的区域级加密密钥，将 DEK 加密成*加密* **DEK**（**encrypted DEK**，**EDEK**）。之后，EDEK 作为指定文件 NameNode 元数据中的扩展属性永久存在。

HDFS 加密区提供了一个能够镜像外部安全域的工具。以一个具有多部门的公司为例，每个部门都需要维护一些仅限本部门可用的数据集。可以通过在每个部门创建一个加密区以保护各部门的数据，从而省去了在认证密钥库（keystore）里为每个文件保存唯一密钥的开支。

如果 EDEK 存储在 HDFS 元数据中，那么 EZK 在哪里存储在哪里呢？这些密钥需要确保安全，因为泄露 EZK 会导致所有存储在该加密区的数据都能够被他人访问。为了防止

Hadoop 管理员访问 EZK 从而能够加密任何数据，EZK 不能存储在 HDFS 中。EZK 需要通过安全的 **key server** 来访问，key server 本身是一个用于处理 EZK 的存储和获取的独立软件。在较大的公司中，实际的存储组件由一个专用的硬件安全模块（**hardware security module**，**HSM**）来处理。通过该部署，key server 就成为了请求密钥的客户端和后端安全存储的软件接口。

为了进行职责分离，需要在 HDFS、HDFS 客户端和 key server 之间有个中间层。Hadoop 密钥管理服务（**Key Management Server**，**KMS**）的引入解决了这个问题。KMS 负责生成加密密钥（包括 EZK 和 DEK）、与 key server 通信以及解密 EDEK。KMS 通过名为 **KeyProvider** 的 Java API，与 key server 进行通信。稍后讲解 KeyProvider 的实现和配置。

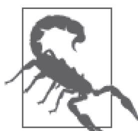
为了更好地理解其工作流程，下面看看当一个 HDFS 客户端向 HDFS 加密区中写入一个新文件时，发生的事件序列。

(1) HDFS 客户端调用 `create()` 函数写新文件。
(2) NameNode 请求 KMS 使用给定的 EZK-id/ 版本创建一个 EDEK。
(3) KMS 生成一个新的 DEK。
(4) KMS 从 key server 获取 EZK。
(5) KMS 对 DEK 进行加密，形成 EDEK。
(6) KMS 将 EDEK 提交给 NameNode。
(7) NameNode 将 EDEK 保存为文件元数据的扩展属性。
(8) NameNode 将 EDEK 提交给 HDFS 客户端。
(9) HDFS 客户端提交 EDEK 到 KMS，并请求 DEK。
(10) KMS 向 key server 请求 EZK。
(11) KMS 使用 EZK 解密 EDEK。
(12) KMS 将 DEK 提交给 HDFS 客户端。
(13) HDFS 客户端使用 DEK 加密数据。
(14) HDFS 客户端向 HDFS 写入加密数据块。

读取加密文件的事件序列如下。

(1) HDFS 调用 `open()` 函数来读文件。
(2) NameNode 将 EDEK 提交给客户端。
(3) HDFS 客户端将 EDEK 和 EZK-id/ 版本传递给 KMS。
(4) KMS 向 key server 请求 EZK。
(5) KMS 使用 EZK 解密 EDEK。
(6) KMS 将 DEK 提交给 HDFS。
(7) HDFS 客户端读取加密数据块，并使用 DEK 对其解密。

读和写的事件中，都没有提到 HDFS 授权。授权检查在文件可以被创建或打开之前仍然是存在的。加密 / 解密步骤仅在 HDFS 授权检查之后进行。



KMS 在 HDFS 加密中扮演着重要角色，因此不应被放在运行着其他 Hadoop 生态系统组件的服务器或者作为客户端边界节点的服务器上。需要有一种合适的职责分离，并且加密的关键操作和其他操作之间需要进行隔离。

由于 KMS 和 key server 以及 HDFS 客户端之间的通信都包含加密密钥的传递，因此使用 TLS（传输层安全协议）对该通信也进行加密绝对是至关重要的。下一节将讨论如何做到这一点。

1. 配置

本章介绍了很多关于 HDFS 加密的内容，但还没有讨论如何对这些内容进行配置。在每个 HDFS 节点和客户端节点的 `core-site.xml` 中设置如下参数。

`hadoop.security.key.provider.path`
: KeyProvider 作为客户端，与加密密钥交互时使用的 URI。示例：`kms://https@kms.example.com:16000/kms`。

在 HDFS 服务端（NameNode 和 DataNode）侧，可以使用如下属性。

`dfs.encryption.key.provider.uri`
: KeyProvider 与读写加密区使用的加密密钥交互时，使用的 URI。示例：`kms:// https@kms.example.com:16000/kms`。

`hadoop.security.crypto.cipher.suite`
: 用于加密编码的加密套件。默认值：`AES/CTR/NoPadding`

`hadoop.security.crypto.codec.classes.aes.ctr.nopadding`
: 由逗号分隔的、AES/CTR/NoPadding 密码编码器的实现方案。第一个实现方案有效时会使用第一个实现方案，其他为备用方案。默认值：`org.apache.hadoop.crypto.OpensslAesCtrCryptoCodec,org.apache.hadoop.crypto.JceAesCtrCryptoCodec`

`hadoop.security.crypto.jce.provider`
: JCE 提供者。默认值：None

`hadoop.security.crypto.buffer.size`
: `CryptoInputStream` 和 `CryptoOutputStream` 使用的缓存大小。默认值：8192

如上所示，HDFS 的配置很少。在密码方面，HDFS 使用合适的默认值，因此要启用 HDFS 加密只需设置前两个配置，即 `hadoop.security.key.provider.path` 和 `dfs.encryption.key.provider.uri`。

要配置 Hadoop KMS，需要用到 `kms-site.xml` 这个配置文件。在 Hadoop KMS 节点上配置如下属性。

`hadoop.kms.key.provider.uri`
: EZK 提供者的 URI。示例：`jceks://file@/var/lib/kms/ kms.keystore`。

`hadoop.kms.authentication.type`
: 要使用的认证机制。示例：simple 或 Kerberos。

`hadoop.kms.authentication.kerberos.keytab`
: 服务认证所需的 Kerberos keytab 文件位置。

`hadoop.kms.authentication.kerberos.principal`
: 服务认证时应当使用的 SPN。示例：`HTTP/kms.example.com@EXAMPLE.COM`。

hadoop.kms.authentication.kerberos.name.rules

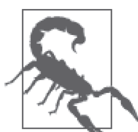
Kerberos 使用的 auth_to_local 规则。示例：DEFAULT。

hadoop.kms.proxyuser.<user>.groups

<user>（如 hdfs、hive、oozie）被允许模拟的组列表。

hadoop.kms.proxyuser.<user>__.hosts++

被允许进行身份模拟的 <user>（如 hdfs、hive、oozie）的来源主机列表。



KMS 配置项中，有一个例子说明了其能够使用基于文件的 KeyProvider。那只是一个保存 EZK 的 Java keystore 文件。虽然这是建立和运行 HDFS 加密的一种快捷简单的方法，但只有在概念验证（POC）或开发环境中才推荐用这种方法进行测试。使用基于文件的 KeyProvider 会将 KMS 和 key server 功能放在同一个机器上，这没有提供要求的职责安全分离，也不具备实施额外的隔离控制的能力。此外，密钥的存储只是磁盘上的一个基础文件。如前所述，多数企业想要使用类似 KeyProvider 的独立服务，它为 EZK 使用更安全的存储方式，例如 HSM 提供的存储。

从 KMS 配置能够看出，可以使用 Kerberos 强认证，这绝对是推荐配置。由于 KMS 提供内容的敏感性，不应使用非 Kerberos 的部署。实际的 KMS 运行在 HTTP 协议之上，因此 KMS 客户端的 Kerberos 认证发生在 SPNEGO 上。所以，KMS 使用的 Kerberos 主体应当与 HTTP/kms.example.com@EXAMPLE.COM 类似——使用 HTTP 服务名。

上一节提到过，为 KMS 设置 TLS 线上加密是很重要的。要做到这一点，需要在 kms-env.sh 中为 KeyStore 和密码配置两个环境变量。KeyStore 文件仅是一个 .jks（Java KeyStore）文件，其位置由环境变量 KMS_SSL_KEYSTORE_fiLE 指定。如果 KeyStore 使用了密码保护（这也是应当做的！），则要在环境变量 KMS_SSL_KEYSTORE_PASS 中指定密码。

2. KMS授权

与其他 Hadoop 组件一样，KMS 也能够使用访问控制表（ACL）限制对特定功能的访问。kms-acls.xml 文件存储了哪些用户和组可以执行哪些 KMS 功能的相关信息，如示例 9-1 所示。

示例 9-1　KMS 的 kms-acls.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>hadoop.kms.blacklist.CREATE</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>hadoop.kms.blacklist.DELETE</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>hadoop.kms.blacklist.ROLLOVER</name>
    <value>hdfs supergroup</value>
  </property>
```

```
  <property>
    <name>hadoop.kms.blacklist.GET</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>hadoop.kms.blacklist.GET_KEYS</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>hadoop.kms.blacklist.SET_KEY_MATERIAL</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>hadoop.kms.blacklist.DECRYPT_EEK</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>default.key.acl.MANAGEMENT</name>
    <value> infosec</value>
  </property>
  <property>
    <name>default.key.acl.GENERATE_EEK</name>
    <value>hdfs supergroup</value>
  </property>
  <property>
    <name>default.key.acl.DECRYPT_EEK</name>
    <value> hadoopusers</value>
  </property>
  <property>
    <name>default.key.acl.READ</name>
    <value> infosec</value>
  </property>
</configuration>
```

示例 9-1 中的每个条目都有一个格式为“用户 1，用户 2 组 1，组 2”的值，与 6.2 节描述的一样。你可能注意到了黑名单的使用。为了将 Hadoop 管理员与实际数据强制隔离，Hadoop 管理员不应能够与 KMS 交互，也不能在 KMS 上执行操作。属于超级组（supergroup）的 Hadoop 管理员有遍历整个 HDFS 目录树的能力，而加密数据不应能被集群管理员解密，因此必须将这些用户加入黑名单。

记住，Hadoop KMS 是个通用的密钥管理服务，它所管理的密钥是没有含义的，或者说对它们的处理方式上是没有区别的。这意味着从 KMS 的角度看，EZK 和 DEK 是一样的。这就是为什么使用 KMS ACL 是很重要的。例如，默认情况下，CREATE 操作会返回实际的密钥内容。如果普通用户能够获取实际的 EZK 那就糟了，因为它可以用于解密 EDEK，从而获取整个加密区的内容。



允许任意用户创建密钥会带来一些潜在的安全风险。例如一个恶意用户可以轻易地写个脚本，不停创建新密钥，直至 KMS 和／或 key server 宕掉（如存储空间不足）。这实际上制造了一个拒绝服务（dos）的场景，导致所有加密数据都无法访问！使用限制性的 KMS ACL 可以授权一小组安全管理员能够创建和管理密钥。

建议（至少）建立 3 个不同的角色以应用 ACL：Hadoop 管理员、安全管理员和普通用户。该模型下，Hadoop 管理员只需能够请求 KMS 生成新 EDEK；安全管理员负责创建和维持 EZK；最后，集群的普通用户只能请求 KMS 解密给定的 EDEK。

继续研究这个模型，示例 9-1 展示了 Hadoop 管理员——hdfs 用户和 supergroup 组——被列入黑名单，从而被禁止执行不必要的操作。此外，infosec 组是唯一一个被允许执行 MANAGEMENT 和 READ 功能的组。最后，hadoopusers 组只被允许执行 DECRYPT_EEK 功能。

示例 9-1 展示的是使用 default.key.acl 前缀标记的默认 ACL，也可以将 ACL 定义为特定名称的密钥，例如 key.acl.foo.READ，其中 foo 是密钥名称。下一节介绍 HDFS 加密客户端操作时，将讨论如何使用密钥名称。

3. 客户端操作

之前从工作流和配置的角度介绍了 HDFS 加密，现在需要了解把这一切都建立起来的客户端操作。首先看新 EZK 的创建。要做到这一点，需要使用 hadoop key 命令，该命令会输出创建的密钥，以及执行该请求的 KMS 的详细信息。

```
[bob@server1 ~]$ hadoop key -create myzonekey
myzonekey has been successfully created with options
 Options{cipher='AES/CTR/NoPadding', bitLength=128, description='null', attr
ibutes=null}.
KMSClientProvider[https://kms.example.com:16000/kms/v1/] has been updated.
[bob@server1 ~]$
```

现在可以在 HDFS 中创建一个新的加密区。使用 hdfs crypto 命令实现：

```
[bob@server1 ~]$ hdfs dfs -mkdir /myzone
[bob@server1 ~]$ hdfs crypto -createZone -keyName myzonekey -path /myzone
[bob@server1 ~]$ hdfs crypto -listZones
/myzone   myzonekey
[bob@server1 ~]$
```

从这开始，HDFS 客户端可以读写 /myzone 目录的文件，并对其进行透明加密 / 解密。



HDFS 加密区的创建需要一个空目录，而不能是一个包含数据的目录。要对已经存在于 HDFS 中的数据加密，要将需要加密的目录重命名为一个临时名称，重新创建改目录，设置加密区，然后将数据复制回加密区。记住，原始数据之前是以未加密的形式存储在磁盘上的，将数据加密不会删除磁盘上原本暴露的未加密敏感数据。

9.2.3 MapReduce2中间数据加密

HDFS 加密启用时，对数据的临时、中间版本进行保护也很重要。虽然对 MapReduce 作业的中间数据进行加密是可行的，但仍有一些注意事项：

- 中间数据加密是基于单个作业的（客户端配置）；
- 用户可能不知道源数据来自加密区；
  — 用户可能没有正确启用中间数据加密
  — 用户可能由于性能影响禁用了中间数据加密

- 中间数据加密仅支持 MR2，不支持 MR1。

表 9-1 中的作业配置属性用于启用中间数据加密。

表9-1：中间数据加密属性

属性	描述
mapreduce.job.encrypted-intermediate-data	设置为 true 以启用（默认值 false）
mapreduce.job.encrypted-intermediate-data-key-size-bits	加密密钥长度（默认：128）
mapreduce.job.encrypted-intermediate-data.buffer.kb	以 kB 为单位的缓存大小（默认值 128）

有实际需要时，开启和强制使用中间数据加密当然是理想的。我们希望后续的 Hadoop 发布版能够实现这项功能，从而使得 MapReduce 任务检测到自己正在读取来自加密区的数据时，能够自动加密中间文件，并且该特性无法被客户端重写。

9.2.4　Impala磁盘溢出加密

Impala 能够在内存无法容纳正在处理的所有数据时，对溢出到磁盘的数据进行加密。如果没有磁盘溢出加密，那么敏感数据有可能会在未加密的情况下被写回磁盘，这会对敏感数据造成危害，使在一开始对数据进行加密的优势毁于一旦。

为了让 Impala 守护进程保护溢出到磁盘的数据，需要配置如下启动标识。

`disk_spill_encryption`

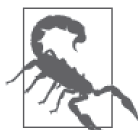
　将该值设为 `true`，开启对队列中溢出到磁盘的所有数据的加密。默认值：`false`。数据即将溢出到磁盘时，它会被使用随机生成的 AES 256 位密钥进行加密；数据被从磁盘中读回时，随之解密。

`disk_spill_integrity`

　将该值设为 `true`，开启对队列中溢出到磁盘的所有数据的完整性检查。默认值：`false`。数据即将溢出到磁盘时，该数据的 SHA256 散列值会被记录下来；从磁盘读回时，会再次计算其 SHA256 值，并与原来的值进行比较。这能够防止溢出到磁盘上的数据被篡改。

9.2.5　全盘加密

如果使用不支持本地加密的 HDFS 版本，或者需要对其他 Hadoop 生态系统组件使用的数据进行加密，那么可以考虑全盘加密或文件系统加密。先看一下使用 **Linux 统一密钥设置**（**Linux Unified Key Setup**，**LUKS**）的全盘加密。除 LUKS 之外，还有一些用于全盘加密的其他产品。我们将重点介绍 LUKS，因为它是在 Linux 上启用全盘加密的开源工具。



生产或真实数据中，不能轻易进行数据加密的试验，因为一个错误可能就会导致数据永久不可恢复。

大多数 LUKS 的实现使用 Linux 发行版中自带的 `cryptsetup` 和 `dm-crypt`：

- `cryptsetup` 提供创建、配置、管理加密卷的用户空间工具
- `dm-crypt` 提供加密块设备的 Linux 内核空间逻辑

示例 9-2 中，我们介绍了如何使用命令行在设备上配置 LUKS。一些 Linux 发行版具有的工具可以在操作系统安装时进行简单配置。建立存储磁盘、增加额外磁盘或对已有磁盘重新分区时，启用全盘加密可以像勾选一个复选框或选择一个选项这么简单。这隐藏了使用 `cryptsetup` 和 `dm-crypt` 的复杂过程。推荐尽可能使用发行版提供的工具。



在设备上建立 LUKS 时，数据会被覆盖。如果在已经有数据的设备上建立 LUKS，首先要对整个设备进行备份，配置完 LUKS 后再恢复数据。进行 LUKS 配置时要谨慎行事。

示例 9-2　LUKS 加密

(1) 安装 `crypsetup`。

在 CentOS/RHEL 中：

```
[root@hadoop01 ~]# yum install cryptsetup-luks
```

On Debian/Ubuntu:

```
[root@hadoop01 ~]# apt-get install cryptsetup
```

(2) 建立 LUKS 存储设备。

```
[root@hadoop01 ~]# cryptsetup -y -v luksFormat /dev/xvdc
WARNING!
========
This will overwrite data on /dev/xvdc irrevocably.

Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
Command successful.
```

(3) 打开设备并将其映射到一个新设备。

```
[root@hadoop01 ~]# cryptsetup luksOpen /dev/xvdc data1
```

这在 /dev/mapper/data1 上创建了一个新的映射设备。

(4) 清除设备上的所有数据（这主要是清除数据头，但清除所有数据是个好的安全方法）。

```
[root@hadoop01 ~]# dd if=/dev/zero of=/dev/mapper/data1
```



前面的操作是在整个存储设备上写 0，因此，根据设备的大小和系统的速度，这会耗费几分钟到几小时。

(5) `dd` 命令完成后，可以创建文件系统；此处使用 `ext4`，但也可以使用 `XFS` 或其他想要的文件系统格式。

```
[root@hadoop01 ~]# mkfs.ext4 /dev/mapper/data1
```

(6) 既然有了一个带文件系统的加密设备，现在可以像普通文件系统一样加载它了。

```
[root@hadoop01 ~]# mkdir /data/dfs/data1
[root@hadoop01 ~]# mount /dev/mapper/data1 /data/dfs/data1
[root@hadoop01 ~]# df -H
[root@hadoop01 ~]# ls -l /data/dfs/data1
```

(7) 对于加载到 `/data/dfs/data[2-N]` 的其他设备，重复前面的步骤，然后使用 `/data/dfs/data[1-N]` 来为 HDFS 存储安装 Hadoop。

这只是一个示例，它没有涵盖如何提供开机密码、如何执行备份等很多其他方面。如果需要增加一个磁盘该怎么办？如果需要调整分区大小呢？在生产部署中，这些都是应该考虑的问题。

9.2.6 文件系统加密

很多产品提供文件系统加密，但我们主要关注 eCryptfs，因为它是 Linux 中一个常见的开源文件系统加密方案。

eCryptfs 有两个主要部分：`ecryptfs-utils` 和 `ecryptfs`。

- `ecryptfs-utils` 提供创建、配置和管理加密目录的用户空间工具
- `ecryptfs` 提供将加密文件系统放到现有文件系统目录之上的 Linux 内核空间逻辑

示例 9-3 将介绍如何使用命令行配置 eCryptfs。一些 Linux 发行版具有的工具能够在操作系统安装时进行简单配置。建立存储磁盘、增加额外磁盘或对已有磁盘重新分区时，可以像勾选一个复选框或选择一个选项这么简单。这隐藏了使用 `ecryptfs-utils` 和 `ecryptfs` 的复杂过程。我们推荐尽可能使用发行版提供的工具。

示例 9-3　eCryptfs 加密

(1) 安装 eCryptfs-utils。

在 CentOS/RHEL 中：

```
[root@hadoop01 ~]# yum install ecryptfs-utils
```

在 Debian/Ubuntu 中：

```
[root@hadoop01 ~]# apt-get install ecryptfs-utils
```

(2) 将一个新的加密文件系统挂载到空的 HDFS 数据目录上。

```
[root@hadoop01 ~]# mount -t ecryptfs /data/dfs/data1 /data/dfs/data1
Select key type to use for newly created files:
1) passphrase
2) tspi
3) openssl
Selection: 1
Passphrase:

Select cipher:
```

```
1) aes: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
2) blowfish: blocksize = 16; min keysize = 16; max keysize = 56 (not loaded)
3) des3_ede: blocksize = 8; min keysize = 24; max keysize = 24 (not loaded)
4) twofish: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
5) cast6: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
6) cast5: blocksize = 8; min keysize = 5; max keysize = 16 (not loaded)
Selection [aes]: aes

Select key bytes:
1) 16
2) 32
3) 24
Selection [16]: 32
Enable plaintext passthrough (y/n) [n]: n
Enable filename encryption (y/n) [n]: n
Attempting to mount with the following options:
ecryptfs_unlink_sigs
ecryptfs_key_bytes=32
ecryptfs_cipher=aes
ecryptfs_sig= 9808e34a098f3814
WARNING: Based on the contents of [/root/.ecryptfs/sig-cache.txt],
it looks like you have never mounted with this key
before. This could mean that you have typed your
passphrase wrong.
Would you like to proceed with the mount (yes/no)? : yes
Would you like to append sig [9808e34a098f3814] to
[/root/.ecryptfs/sig-cache.txt]
in order to avoid this warning in the future (yes/no)? : yes
Successfully appended new sig to user sig cache file
Mounted eCryptfs
```



mount 命令过程中，你会被要求输入以字节为单位的密钥大小。之前我们将期望的密钥大小设为 256 位，由于 1 字节包含 8 位，因此此处选择 32 位密钥。

(3) 对于加载到 `/data/dfs/data[2-N]` 的其他设备，重复前面的步骤，然后使用 `/data/dfs/data[1-N]` 为 HDFS 存储安装 Hadoop。

这个例子没有涵盖如何提供开机密码、如何进行备份或者忘记密码时怎么办等其他方面。这些是在大规模使用加密方案时需要额外考虑的

9.2.7 Hadoop中重要数据的安全考虑

如果在为 Hadoop 的静态数据配置加密，需要注意，敏感数据不仅存在于 HDFS 上，也存在于运行在 MySQL、PostgreSQL、SQLite、Oracle 或 Derby 的 shuffle、溢出填充、临时文件、日志文件、交换文件、索引和元数据库等其他地方。第 10 章将介绍 Hadoop 用于提供 HDFS 之外的数据集加密的一些地方。

9.3 动态数据加密

上一节描述了如何使用加密保护静态数据，但数据正在网络上传输时，该怎么办呢？先举一个抽象的例子。当 Alice 和 Bob 还是孩子的时候，他们喜欢在课堂上传纸条。由于他们不是总坐在一起，因此需要依赖其他孩子帮忙传递。他们怎么对付那些可能想在传走纸条之前读一下纸条内容的爱管闲事的小孩呢？或者更糟的是，如果被老师抓到并且向全班同学读出纸条内容会怎样呢？

作为一个聪明的孩子，Alice 想出一个她自己的字母表，字母表中包括一一映射英语字母的符号。他们不用英语字母，而是用这个自定义的字母表写自己的消息。Alice 和 Bob 可以提前交换一份映射关系，甚至记下这个字母表（毕竟只有 26 个符号）。现在，当他们传出自己的纸条时，只有拥有那个字母表副本的人才能读懂内容。

这种方法简单有效，但不是绝对安全的。更复杂的方法可能包括使用多个字母表或随机映射的字母表，以及一个告诉接收者该使用哪个映射的密钥。这对于传纸条来说可能是小题大做，但设计动态数据加密系统时，这很重要。

9.3.1 传输层安全

传输层安全（**Transport Layer Security**，**TLS**）是一个用于动态数据加密的安全协议。TLS 替代了*安全套接字层*（**Secure Socket Layer**，**SSL**），SSL 是动态数据加密的一个早期标准。TLS 的初次定义是在基于 SSL 3.0 协议的 RFC 2246 中，它是由 Paul Kocher 设计的。由于 TLS 和 SSL 拥有相同的历史，所以虽然这两个协议不同，但它们总被交替使用。使用相同的库实现 SSL 和 TLS 也很常见。例如，OpenSSL 库包含 SSL 2.0 和 SSL 3.0 的实现，也包含 TLS 1.0、1.1 和 1.2 的实现。

第 4 章是启用强认证的协议，而 SSL/TLS 协议保护数据在网络传输时的安全。虽然最常见的是，SSL/TLS 以 HTTPS 协议的形式与 Web 流量联系在一起，但它其实是可用于保护任意套接字连接的通用协议。这使得你可以创建一个加密管道，其他协议可以放在该管道之上。就像 Kerberos 客户端信赖 KDC 一样，使用 SSL/TLS 的客户端信赖中心的*证书颁发机构*（**certificate authority**，**CA**）。

以下是支撑 SSL/TLS 的一些基本概念。

私钥（**Private key**）
: 只有签名证书所有者知道的非对称加密密钥。

公钥（**Public key**）
: 公开的、可用于加密数据的非对称加密密钥，加密的数据只能被对应的私钥解密。

证书签名请求（**CSR**）
: 发送到证书颁发机构以申请特定身份的加密消息。

签名证书
: 发送 CSR 到 CA 的结果。签名证书包括与私钥一起生成的公钥的一个副本。使用自签名证书而不是 CA 签名的证书也可以，但只在测试和开发系统中才推荐这么用。

PKCS #12

包含私钥和签名证书的一种文件格式。

虽然此处不讨论 SSL/TLS 的很多技术细节，但接下来的基本工作流程示例会介绍一些应当了解的内容。

1. 生成新证书

(1) 想要接受 SSL/TLS 连接的服务的管理员生成一个公私钥对。
(2) 管理员之后生成一个 CSR 并发送到 CA。
(3) CA 验证服务器／服务（有时候是企业实体）的身份，然后生产一个签名证书。
(4) 服务的管理员之后可以安装签名证书。

2. SSL/TLS握手

(1) Alice 连接到 Bob 服务，Bob 服务向 Alice 展示一个 SSL/TLS 证书。
(2) Alice 在它信任的第三方链中查找签发 Bob 的证书的 CA 证书。
(3) Alice 和 Bob 服务交换公钥，然后为当前会话协商一个新创建的对称密钥。
(4) Alice 向 Bob 发送一条信息，该信息使用通过安全交换得来的对称密钥进行动态加密。
(5) 即使 Even 捕获到从 Alice 到 Bob 的消息数据包，她也无法对其解密，因为她没有对称密钥。

这种设计的一个众所周知的实现就是 RSA 密钥交换算法。该算法中，紧跟着私钥和公钥对生成之后的是公钥的安全交换，它允许通信双方发送只能被指定接收者解密的加密消息。



RSA 的名字由 Ron Rivest、Adi Shamir 和 Leonard Adlemanwho 的姓氏而来，他们 1977 年在 MIT 的时候写了关于该算法的一篇论文。在本书中你可以看到，除了 Kerberos 之外，还有很多我们现在使用的安全技术都源于 MIT。

要想更深入地了解 SSL/TLS，推荐阅读 John Viega、Matt Messier 和 Pravir Chandra 编著的图书 *Network Security with OpenSSL*，以及 Scott Oaks 编著的 *Java Security*(Second Edition)。

9.3.2 Hadoop动态数据加密

Hadoop 有多种网络通信方式，包括 RPC、TCP/IP 和 HTTP。MapReduce、JobTracker、TaskTracker、NameNode 和 DataNode 的 API 客户端均使用 RPC 调用。HDFS 客户端使用 TCP/IP 套接字进行数据传输。HTTP 协议用于 MapReduce shuffle，很多 Web UI 的守护进程也使用 HTTP 协议。

这三种网络通信每种都有不同的动态加密方法。接下来探讨这几种加密方法的基础知识，第 10 章会介绍一个 Flume SSL/TLS 配置的详细示例，第 11 章和第 12 章还会介绍 Oozie、HBase、Impala 和 Hue 中 SSL/TLS 的使用。

1. Hadoop RPC加密

Hadoop 的 RPC 实现支持 SASL，SASL 除了支持身份认证外，还提供了可选的信息完整性和信息加密。Hadoop 使用 Java 的 SASL 实现，支持以下几种模式。

- auth：用于客户端和服务端之间的身份验证
- auth-int：用于身份验证和完整性
- auth-conf：用于身份验证、完整性和机密性

Hadoop 中的 RPC 保护在 core-site.xml 文件中的 hadoop.rpc.protection 属性中配置。该属性可以被设置成如下值。

- authentication：默认值，将 SASL 设置为 auth 模式，仅提供身份验证。
- integrity：将 SASL 设置为 auth-int 模式，在身份验证之外增加完整性验证。
- privacy：将 SASL 设置为 auth-conf 模式，增加加密以保证完全的机密性。

配置 Hadoop RPC 保护，需要在 core-site.xml 中按照如下所示进行设置。（记住，需要重启所有守护进程才能使该设置生效。）

```
<property>
  <name>hadoop.rpc.protection</name>
  <value>privacy</value>
</property>
```

2. HDFS数据传输协议加密

HDFS 数据从一个 DataNode 传输到另一个 DataNode 时，或者在 DataNode 和客户端之间传输时，会使用一个名为 **HDFS 数据传输协议**（**HDFS data transfer protocol**）的 TCP/IP 直连套接字。数据传输加密启用时，会使用 Hadoop RPC 协议交换数据传输协议中使用的加密密钥。

配置数据传输加密需要在 hdfs-site.xml 文件中，将 dfs.encrypt.data.transfer 设置为 true——只有在 DataNode 上才要求进行此更改。RPC 会用于交换加密密钥，因此要设置 hadoop.rpc.protection 配置项为 privacy，以保证启用 RPC 加密，如前所述。加密算法也需要被配置成使用 AES。通过下面的代码配置 AES 加密：

```
<property>
  <name>dfs.encrypt.data.transfer</name>
  <value>true</value>
</property>
<property>
  <name>dfs.encrypt.data.transfer.cipher.suites</name>
  <value>AES/CTR/NoPadding</value>
</property>
<property>
  <name> dfs.encrypt.data.transfer.cipher.key.bitlength</name>
  <value>256</value> <!-- can also be set to 128 or 192 -->
</property>
```



使用 dfs.encrypt.data.trans fer.cipher.suites 设置 AES 加密是 2.6 版本加入的一个较新的 Hadoop 功能。对于之前的版本，可以设置 dfs.encrypt.data.transfer.algorithm 为 3des（默认值）或 rc4，以分别选择 triple-DES 或 RC4。

需要重启 DataNode 和 NameNode 守护进程，使设置生效。整个过程可以手动完成，Hadoop 发行版可能也会提供自动化的方法启用 HDFS 数据传输加密。

3. Hadoop HTTP加密

谈到 HTTP 加密，有一个众所周知的成熟方法——使用 **HTTPS** 加密传输中的数据。HTTPS 是一个使用 SSL/TLS 的 HTTP 增强。虽然 HTTPS 是很标准化的，但在 Hadoop 中对它进行配置并非如此。一些 Hadoop 组件支持 HTTPS，但它们的配置步骤并不都一样。

你可能会回忆起我们描述基本的 SSL/TLS 概念时，提到需要一些额外的文件，例如私钥、证书和 PKCS #12 包。使用 Java 时，这些文件存储在 Java **keystore** 里。

一些 Hadoop 组件对于其他服务来说既是 HTTPS 服务端也是客户端。下面是一些例子：

- HDFS、MapReduce 和 YARN 守护进程既作为 SSL 服务端也作为 SSL 客户端
- HBase 守护进程只作为 SSL 服务端
- Oozie 守护进程只作为 SSL 服务端
- Hue 作为以上所有的 SSL 客户端

我们不深入讨论 HTTPS 配置，而重点关注 MapReduce 加密 shuffle 和加密 Web UI 的配置，作为配置其他组件的出发点。

4. 加密shuffle和加密web Web UI

MR1 和 MR2 都支持加密 shuffle。在 MR1 中，设置 `core-site.xml` 文件中的 `hadoop.ssl.enabled` 属性可以启用加密 shuffle 和加密 Web UI。在 MR2 中，设置 `hadoop.ssl.enabled` 属性只能启用加密 Web UI 的功能，设置 `mapred-site.xml` 文件中的 `mapreduce.shuffle.ssl.enabled` 属性能够启用加密 shuffle 的功能。



与 Kerberos 一样，配置 HTTPS 时一个很重要的事情是，要使用完整的主机名配置你的服务器；并配置 DNS，使其能够在集群中正确解析这些主机名。

在 MR1 和 MR2 中，设置 `core-site.xml` 中的 `ssl` 属性都能启用加密 Web UI。在 MR1 中，该设置还会启用加密 shuffle。

```
<property>
  <name>hadoop.ssl.enabled</name>
  <value>true</value>
  <final>true</final>
</property>
```

仅对于 MR2，要在 **mapred-site.xml** 中设置加密 shuffle SSL 属性。

```
<property>
  <name>mapreduce.shuffle.ssl.enabled</name>
  <value>true</value>
  <final>true</final>
</property>
```

作为一个可选项，也可以设置 `hadoop.ssl.hostname.verifier` 属性控制如何进行主机名验证。可选的值如下所示。

DEFAULT

主机名必须与第一个 CN 或任意 SAN（subject-alt name）匹配。如果 CN 或某个 SAN 中存在通配符，则匹配所有子域名。

DEFAULT_AND_LOCALHOST

与 DEFAULT 模式的作用一样，添加 localhost 主机，localhost.localdomain、127.0.0.1 和 ::1 总会通过主机名验证。

STRICT

与 DEFAULT 模式的作用相似，但只在同级匹配通配符。例如 *.example.com 能够匹配 one.example.com，但不能匹配 two.one.example.com。

ALLOW_ALL

接受任意主机名。该模式不安全，应当仅在测试中使用。

例如要支持 default 加上 localhost 模式，则按如下所示进行配置：

```
<property>
  <name>hadoop.ssl.hostname.verifier</name>
  <value>DEFAULT_AND_LOCALHOST</value>
  <final>true</final>
</property>
```

还需要更新 ssl-server.xml 和 ssl-client.xml 文件，这些文件通常在 /etc/hadoop/conf 目录下。ssl- server.xml 里的设置如表 9-2 所示。

表9-2：ssl-server.xml的Keystore和Truststore设置

属性	默认值	描述
ssl.server.keystore.type	jks	keystore 文件类型
ssl.server.keystore.location	NONE	keystore 文件路径；该文件应当为 mapred 用户所有，mapred 用户必须拥有对其的独占读权限（即 400 权限）
ssl.server.keystore.password	NONE	keystore 文件的密码
ssl.server.truststore.type	jks	truststore 文件类型
ssl.server.truststore.location	NONE	truststore 文件路径；该文件应当为 mapred 用户所有，mapred 用户拥有对其的独占读权限（即 400）
ssl.server.truststore.password	NONE	truststore 文件密码
ssl.server.truststore.reload.interval	10000	truststore 文件刷新间隔的毫秒数

一个完整配置的 **ssl-server.xml** 示例如下：

```
<configuration>
  <!-- Server keystore -->
  <property>
    <name>ssl.server.keystore.type</name>
    <value>jks</value>
  </property>
  <property>
    <name>ssl.server.keystore.location</name>
    <value>/etc/hadoop/ssl/server/hadoop01.example.com.jks</value>
```

```
    </property>
    <property>
      <name>ssl.server.keystore.password</name>
      <value>super-secret-squirrel</value>
    </property>

    <!-- Server truststore -->
    <property>
      <name>ssl.server.truststore.type</name>
      <value>jks</value>
    </property>
    <property>
      <name>ssl.server.truststore.location</name>
      <value>/etc/hadoop/ssl/server/truststore.jks</value>
    </property>
    <property>
      <name>ssl.server.truststore.password</name>
      <value>changeit</value>
    </property>
    <property>
      <name>ssl.server.truststore.reload.interval</name>
      <value>10000</value>
    </property>
  </configuration>
```

`ssl-client.xml` 文件中的配置见表 9-3。

表9-3：ssl-client.xml中的keystore和truststore设置

属性	默认值	描述
`ssl.client.keystore.type`	`jks`	keystore 文件类型
`ssl.client.keystore.location`	`NONE`	keystore 文件路径；该文件应当为 mapred 用户所有，所有可以运行 MapReduce 作业的用户应当具有对其的读权限（即 444 权限）
`ssl.client.keystore.password`	`NONE`	keystore 文件密码
`ssl.client.truststore.type`	`jks`	truststore 文件类型
`ssl.client.truststore.location`	`NONE`	truststore 文件路径；该文件应当为 mapred 用户所有，所有可以运行 MapReduce 作业的用户应当具有对其的读权限（即 444 权限）
`ssl.client.truststore.password`	`NONE`	truststore 文件密码
`ssl.client.truststore.reload.interval`	`10000`	truststore 文件刷新间隔的毫秒数

一个完整配置的 **ssl-client.xml** 文件如下所示：

```
  <configuration>
    <!-- Client keystore -->
    <property>
      <name>ssl.client.keystore.type</name>
      <value>jks</value>
    </property>
    <property>
      <name>ssl.client.keystore.location</name>
```

```
    <value>/etc/hadoop/ssl/client/hadoop01.example.com.jks</value>
  </property>
  <property>
    <name>ssl.client.keystore.password</name>
    <value>super-secret-squirrel</value>
  </property>

  <!-- Client truststore -->
  <property>
    <name>ssl.client.truststore.type</name>
    <value>jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.location</name>
    <value>/etc/hadoop/ssl/client/truststore.jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.password</name>
    <value>changeit</value>
  </property>
  <property>
    <name>ssl.client.truststore.reload.interval</name>
    <value>10000</value>
  </property>
</configuration>
```



配置 SSL/TLS 时，一定要确保禁用明文服务。若一个服务运行在 HTTP 80 端口，又配置了 HTTPS 并运行在 443 端口，那么一定要禁用运行在 80 端口上的 HTTP。为了更强的保护级别，可以配置一个防火墙（如 `iptables` 软件防火墙）禁用对 80 端口的访问。

设置 `ssl-server.xml` 和 `ssl-client.xml` 文件后，需要重启 MR1 中的 TaskTracker 和 MR2 中的 NodeManager，使改动生效。

9.4 数据销毁和删除

涉及数据安全时，如何删除数据是很重要的一方面。如果用户恰好要重复使用集群中先前被用于处理敏感数据的服务器，首先需要销毁这些敏感数据。如示例 9-2 所示，使用 dd 命令将 LUKS 分区清零。

使用 GNU `shred` 工具能够对数据进行更彻底的销毁。`shred` 工具会用随机模式对文件或设备进行覆写，从而更好地混淆之前写入的数据。用户可以向 `shred` 传递大量迭代式，默认的迭代次数为 3 次。旧版 DoD 5220.22-M 标准强制要求进行 7 次覆写才能安全清除敏感数据。最安全的覆写模式是 Gutmann 方法，该方法要求使用随机数据和特定数据结合的模式进行 35 次覆写。

在大容量磁盘上进行 35 次覆写是一项非常耗时的工作。假设硬盘可以恒定 100 MB/s 的速度写数据，想要完全覆写一块 2 TB 的磁盘 35 次，将会超过 200 小时，大概相当于 8.5 天。

处理一个拥有数以百计的机器和上千上万块磁盘的集群时，即使执行并行化清除，也是一项艰巨的任务。

DoD 标准在近年来已经发展到，认为对于特别敏感的数据，做再多的覆写操作也不够。这种情况下，只有消磁或物理销毁是可接受的。还有很重要的一点是，被损坏的磁盘不能被覆写，因此只能进行消磁或物理销毁。磁盘故障很常见且数据足够敏感的大集群里，应当有合适的物理销毁计划。许多设备能对磁盘进行物理销毁，还有一些公司提供此类现场服务，他们会将粉碎设备带至现场。

9.5 小结

本章讨论了如何使用加密技术保护数据不受用户和 Hadoop 集群管理员的非授权访问。我们对比了保护静态数据和动态数据的区别，阐述了 HDFS 最近添加的本地静态数据加密和适用于早期版本的替代方法，以及用于 HDFS 外部静态数据加密的方法。还展示了数据处理或查询过程中生成的中间数据如何被加密，以进行端到端保护。

接下来，我们讨论了使用 Hadoop RPC 加密手段对动态数据进行保护的方法，继续通过学习 HDFS 数据传输协议加密的形式学习保护从 HDFS 客户端到 DataNode 以及多个 DataNode 之间的数据。我们还讨论了如何加密 HTTP 端点以及进行带 SSL/TLS 的 MapReduce shuffle。最后，介绍了永久销毁数据的方法，说明如何将对数据的保护延伸到硬件的整个使用周期。

接下来的两章将会分别探索将数据安全扩展至数据导入管道和客户端访问，从而全面保护 Hadoop 环境。

第 10 章

数据导入安全

前面的章节着重从存储和数据处理的角度讲解了 Hadoop 安全。我们假设你在 Hadoop 中已存有数据，且想要保护对这些数据的访问；或者你想要控制用户如何分享已有的分析资源，但还没有解释数据起初是如何被导入 Hadoop 的。

有许多种方法能够将数据导入 Hadoop。最简单的途径是使用 Hadoop 的 `put` 命令，从本地文件系统（如本地硬盘或挂载的 NFS）将文件复制到 HDFS，如示例 10-1 所示。

示例 10-1　从命令行导入文件

```
[alice@hadoop01 ~]$ hdfs dfs -put /mnt/data/sea*.json /data/raw/sea_fire_911/
```

这种方法适用于某些数据集，但更常见的做法是，从已存在的关系型系统导入数据，或设置面向事件——或日志——的数据流。对于这些用例，用户会分别使用 Sqoop 和 Flume。

Sqoop 被设计为可从关系型数据库拉取数据到 Hadoop，或从 Hadoop 将数据推送到远程数据库。这两种情况下，Sqoop 会运行一个 MapReduce 作业，执行实际的数据传输。Sqoop 默认使用 JDBC 驱动，在映射任务和数据库之间传输数据，这称为*通用模式*。通过该模式，使用 Sqoop 对新数据进行存储将变得更容易，唯一的要求是 JDBC 驱动必须可用。出于性能方面的原因，Sqoop 还支持使用厂商的定制工具和接口的连接器以优化数据传输。要使用这些优化手段，用户可通过配置 `--direct` 选项启用*直接*（**direct**）*模式*。例如，对 MySQL 启用直接模式时，Sqoop 会使用 `mysqldump` 和 `mysqlimport` 工具以更高效地从 MySQL 抽取数据，或将数据导入 MySQL。

Flume 是一个分布式服务，能有效收集、聚合以及移动大量事件数据，Flume 用户需要部署代理，这些代理是用于传输数据的 Java 进程。一个事件是流经 Flume 的数据的最小单元，事件拥有一个二进制（字节数组）的载荷，和一个可选的、由字符串键 / 值对定义的属性集。每一个代理都配置有源模块、阱模块和通道。源模块是能够使用外部数据源的组

件，它既能从外部数据源拉取事件，也能接受从外部数据源推送来的事件。Flume 源会连接到一个通道，该通道使得阱模块能够使用这些来自源模块的事件。这个通道是完全被动的，它从源模块接受数据并一直保存数据，直到阱模块使用掉数据为止。一个 Flume 阱能将事件传输至外部数据存储或进程。

Flume 包括使用 Avro RPC（远程过程调用）的 AvroSource（Avro 源）和 AvroSink（Avro 阱）以传输事件。为了构建复杂的、分布式的数据流，可对某一个 Flume 代理的 AvroSink 进行配置，使之向另一个 Flume 代理的 AvroSource 发送事件数据。虽然 Flume 能够支持多种源模块和阱模块，但实现 agent 间数据传输功能的主要还是 AvroSource 和 AcroSink。因此，接下来的讨论内容将限定在它们之间。Flume 的可靠性由通道的配置决定。对于内存中的数据流通道，速度的优先级高于稳定性，而位于磁盘上的通道则支持完全的可恢复性。图 10-1 展示了一个双层 Flume 数据流的内部组件，包括代理和它们之间的连接。

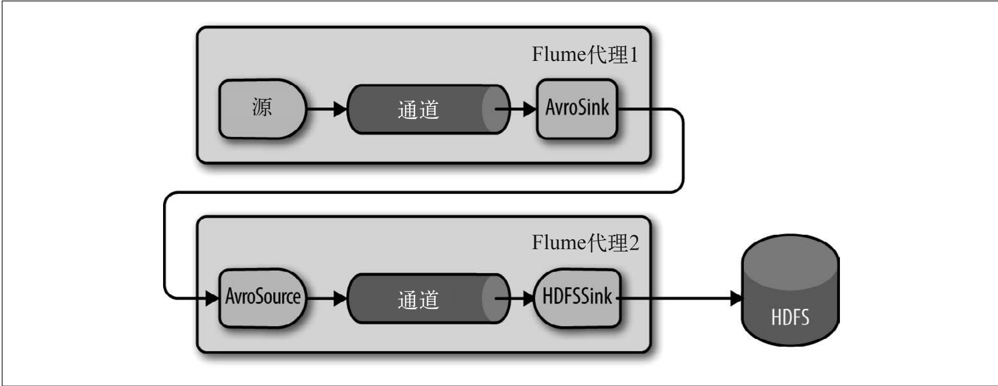


图 10-1：Flume 架构

因为 Sqoop 和 Flume 可能被用于传输敏感数据，所以考虑导入管道在整体部署中的安全性是很重要的，尤其需要关注导入管道的机密性、完整性和可用性（CIA）。机密性是指，将对数据的访问限制在一个拥有授权的用户集之中，通常系统会将身份验证、授权和加密结合在一起，以保证机密性的实现。完整性是指，用户能在多大程度上信任这些数据没有被篡改，大部分系统使用校验和或签名的方式验证数据完整性。可用性是指，保持信息资源的可使用性，在数据导入场景中，可用性意味着导入系统面对一些容量损失时具有鲁棒性。也就是说，系统处理一些下行系统或服务的突然中断时，具有保存传输数据的能力。

10.1　导入数据的完整性

分析过程的价值与数据的价值直接相关，而数据只有在可信时才具有价值，因此，数据的完整性是关键。Hadoop 是复杂的分布式系统，所以毫不意外地，数据导入流也经常具有同等的复杂性和分布性。这意味着数据能在许多地方被使用、损坏和篡改。你的具体威胁模型将决定导入管道所需的完整性的等级。大部分用例都会关注意外损坏的数据，而对于这些情况，对文件或记录的一个简单的校验和就足够了。为了防止数据被恶意用户篡改，可以添加加密签名和 / 或加密记录。

Flume 确保数据完整性的一个主要方法是，使用其内建的、可靠的通道实例。Flume 通道提供一个非常简单的接口，类似于一个无限长队列。通道使用 `put(Event event)` 方法将一个事件导入一个通道，使用 `take()` 方法将下一事件从通道取出。默认的通道实现是一个内存通道，当且仅当 Flume 的代理保持运行时，该实现才是可靠的。这意味着，进程或服务崩溃时，数据就会丢失。此外，由于事件始终不离开内存，Flume 会假定事件不被篡改，因而并不计算或验证事件的校验和。

对于那些关注数据的可靠传输和完整性的用户，Flume 提供了一种基于文件的通道。文件通道本质上实现了一个预写日志，每个事件被导入通道时，该日志被持久地存储到稳定的存储器。除了将事件存入磁盘之外，文件通道还为每个事件计算一个校验和，并将该校验和与事件一起写入预写日志。事件从通道中被取出时，它的校验和会被检验，以确保该事件没有被损坏。这提供了一定程度的完整性保障，然而这种保障对于经过通道传输的事件完整性来说仍然比较局限。目前，Flume 将事件从 AvroSink 的一个代理传递给 AvroSource 的另一个代理时，并不计算校验和。TCP 协议会保护数据包不受意外损坏，然而一个能够控制数据包的中间人仍能以不会被检测到的方式破坏数据。将了解到，Flume 没有能力加密 RPC 协议，而加密 RPC 恰好能够防止未被检测的中间人攻击。

继续讨论 Sqoop 提供的完整性之前，快速了解一下 Flume 如何实现可用性。Flume 允许用户创建一个能够保证“至少发送一次”消息处理语义的分布式数据流。严格地讲，只要第一个代理能够与外部的源通信，Flume 就能够与特定数据源进行通信。事件在代理间被依次处理时，任何下游代理的停机时间都可以通过使用一个故障切换阱处理器进行处理，该处理器以下游的两个或两个以上的 Flume 代理为目标。也可以使用负载均衡的阱处理器同时进行故障处理和负载均衡。这种处理器将会把事件循环或随机地发送到下游阱节点的集合上，如果下游的阱节点出现故障，那么该处理器会尝试再下一个阱节点。

这两种机制都改善了整体数据流的可用性，但它们都不能保证任何某个特定事件的可用性。有人建议可增加一个复制通道，在告知源节点收到数据前，将事件复制到多个代理。但除非这些复制已经完成，否则事件在 Flume 节点宕机时仍会不可用。除非稳定存储文件的通道日志数据不可恢复，否则事件是不会丢失的。创建一个导入流时，记住这些注意事项很重要。我们将在 10.4 节进一步了解这些权衡措施。

与 Flume 不同，Sqoop 本身不支持验证导入数据的完整性。Sqoop 的优势在于，它可以利用同一个向 HDFS 写数据的进程从数据库拉取数据，这意味着，发生数据损坏的概率要比复杂的 Flume 流相对小一些。正如下一节将讲解的，可以通过开启 SSL 加密保证 Sqoop 的数据机密性。这同样也提高了破坏传输中数据的难度，从而改善数据完整性，然而并不能帮助验证写入 HDFS 的数据与数据库中数据是否一致。当前验证 Sqoop 导入的主流方法是往返验证表——从数据库到 Hadoop，再返回数据库。可以通过计算原始表的校验和与往返表的校验和验证数据没有丢失。遗憾的是，这个过程开销很高，并且可能需要进行多次全表扫描，具体扫描次数取决于数据库的校验性能。

10.2 数据导入的机密性

导入数据流的机密性等级依赖于导入的数据类型、关注的威胁模型和任何可能遵循的监管

要求。数据被导入时，在网络中的传输过程以及在中间服务器的存储容易受到非授权用户的访问。甚至在一个受信任的合作方网络中，为了防止非授权用户有机会嗅探网络流量并获取敏感数据，某些数据必须始终被加密。同样地，数据传输所经服务器的管理员可能也没有权限访问某些被导入的数据。为了防止这些情况下的非授权访问，应当将数据在到达稳定的存储之前进行加密。

10.2.1 Flume加密

为了解决数据在网络传输过程中的非授权访问问题，Flume 支持在 AvroSource 和 AvroSink 上启用 SSL 加密。除了提供加密功能外，还可以给 AvroSource 和 AvroSink 配置信任策略，确保阱只将数据发往一个受信任的源。假设需要从一个运行在 `flume01.example.com` 上的 Flume 代理向运行在 `flume02.example.com` 上的代理发送事件，首先需要使用 `openssl` 命令行工具为 flume02 创建一个 RSA 私钥，如示例 10-2 所示。

示例 10-2　创建一个私钥

```
[alice@flume02 ~]$ mkdir certs
[alice@flume02 ~]$ cd certs
[alice@flume02 certs]$ openssl genrsa -des3 -out flume02.key 1024
Generating RSA private key, 1024 bit long modulus
..............................................................................
........++++++
.....................++++++
e is 65537 (0x10001)
Enter pass phrase for flume02.key:
Verifying - Enter pass phrase for flume02.key:
[alice@flume02 certs]$
```

示例 10-3 中，生成一个证书签名请求，因而会有一个证书被分发给刚刚创建的私钥。

示例 10-3　创建证书签名请求

```
[alice@flume02 certs]$ openssl req -new -key flume02.key -out flume02.csr
Enter pass phrase for flume02.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
---
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:California
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:Cluster, Inc.
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:flume02.example.com
Email Address []:admin@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
```

```
An optional company name []:
[alice@flume02 certs]$
```

一旦有了证书签名请求，就可以生成一个由受信任密钥签名的证书。示例中，没有根签名的权限，所以只生成一个自签名证书（给证书签名的密钥与请求证书的密钥相同）。实际部署中，应当将证书签名请求发送至公司的签名认证机构，他们会提供完成签名的证书。自签名证书如示例 10-4 所示。

示例 10-4　创建一个自签名证书

```
[alice@flume02 certs]$ openssl x509 -req -days 365 -in flume02.csr \
  -signkey flume02.key -out flume02.crt
Signature ok
subject=/C=US/ST=California/L=San Francisco/O=Cluster, Inc./CN=flume02.cluster.
com/emailAddress=admin@example.com
Getting Private key
Enter pass phrase for flume02.key:
[alice@flume02 certs]$
```

我们将在 flume02 上使用刚刚创建的密钥和证书配置 AvroSink，但首先需要创建一个 truststore 供 flume01 验证密钥的真实性。实际部署中，truststore 可以由 CA（认证中心）或 Flume 集群的下属 CA 在进行签名时一起加载。这次将使用 Java 的 keytool，将证书导入 Java 的 truststore。

示例 10-5　创建一个 Java truststore

```
[alice@flume02 certs]$ keytool -import -alias flume02.example.com \
 -file flume02.crt -keystore flume.truststore
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=admin@example.com, CN=flume02.example.com, O="Cluster, Inc.
", L=San Francisco, ST=California, C=US
Issuer: EMAILADDRESS=admin@example.com, CN=flume02.example.com, O="Cluster, Inc
", L=San Francisco, ST=California, C=US
Serial number: 86a6cb314f86328b
Valid from: Tue Jun 24 11:31:50 PDT 2014 until: Wed Jun 24 11:31:50 PDT 2015
Certificate fingerprints:
         MD5:  B6:4A:A7:98:9B:60:3F:A2:5E:0B:BA:BA:12:B4:8D:68
         SHA1: AB:F4:AB:B3:2D:E1:AF:71:28:8B:60:54:2D:C1:C9:A8:73:18:92:31
         SHA256: B1:DD:C9:1D:AD:57:FF:47:28:D9:7F:A8:A3:DF:9C:BE:30:C1:49:CD:85:D3:
95:AD:95:36:DC:40:4C:72:15:AB
         Signature algorithm name: SHA1withRSA
         Version: 1
Trust this certificate? [no]: yes
Certificate was added to keystore
[alice@flume02 certs]$
```

将认证和密钥用于 Flume 之前，需要将它们加载为一个 Java 可读的文件格式，通常会是 Java keystore 的 `.jks` 文件或 PCKS12 的 `.p12`。由于 Java 的 keytool 不支持密钥和证书的分开导入，使用 `openssl` 生成 PKCS12 文件并配置 Flume 直接使用该文件，如示例 10-6 所示。

示例 10-6　使用密钥和证书创建一个 PKCS12 文件

```
[alice@flume02 certs]$ openssl pkcs12 -export -in flume02.crt \
  -inkey flume02.key -out flume02.p12 -name flume02.example.com
Enter pass phrase for flume02.key:
Enter Export Password:
Verifying - Enter Export Password:
[alice@flume02 certs]$
```

配置 Flume 使用证书之前，需要将 PKCS12 文件移至 Flume 的配置目录，如示例 10-7 所示。

示例 10-7　复制 PKCS12 文件至 /etc/flume-ng/ssl 目录

```
[root@flume02 ~]# mkdir /etc/flume-ng/ssl
[root@flume02 ~]# cp ~alice/certs/flume02.p12 /etc/flume-ng/ssl
[root@flume02 ~]# chown -R root:flume /etc/flume-ng/ssl
[root@flume02 ~]# chmod 750 /etc/flume-ng/ssl
[root@flume02 ~]# chmod 640 /etc/flume-ng/ssl/flume02.p12
```

示例 10-8 中，需要将 truststore 复制到 `flume01.example.com` 上，这样阱会知道它能够信任位于 `flume02.example.com` 上的源。

示例 10-8　将 truststore 复制到 flume01.example.com

```
[root@flume02 ~]# scp ~alice/certs/flume.truststore flume01.example.com:/tmp/
```

接下来，示例 10-9 中，将 truststore 转移至 Flume 的配置目录。

示例 10-9　转移 truststore 至 /etc/flume-ng/ssl

```
[root@flume01 ~]# mkdir /etc/flume-ng/ssl
[root@flume01 ~]# mv /tmp/flume.truststore /etc/flume-ng/ssl
[root@flume01 ~]# chown -R root:flume /etc/flume-ng/ssl
[root@flume01 ~]# chmod 750 /etc/flume-ng/ssl
[root@flume01 ~]# chmod 640 /etc/flume-ng/ssl/flume.truststore
```

现在，PKCS12 和 truststore 文件都已到位，可以配置 Flume 的源和阱，首先从 `flume01.example.com` 上的阱开始。关键配置参数如下所示。

`ssl`

　启用 SSL 时设为 `true`。启用 SSL 时，还需要对 `trust-all-certs`、`truststore`、`truststore-password` 和 `truststore-type` 等参数进行配置。

`trust-all-certs`

　要禁用证书验证时，将其设为 `true`。强烈推荐将该参数设为 `false`，因为这样能确保阱节点会检查与其连接的源确实正在使用一个受信任的证书。

`truststore`

　将其设置为 Java truststore 文件的绝对路径。如果为空，Flume 将会使用默认的 Java 认证中心文件。Oracle JRE 装载有 `$JAVA_HOME/jre/lib/security/cacerts` 文件，Flume 将使用该文件，除非在 `$JAVA_HOME/jre/lib/security/jssecacerts` 创建一个指定位置的 truststore 文件。

truststore-password

将该参数设为保护 truststore 的口令。

truststore-type

将该参数设为 JKS 或另一种受支持的 truststore 类型。

具体配置如示例 10-10 所示。

示例 10-10　Avro SSL 阱的配置

```
a1.sinks = s1
a1.channels = c1
a1.sinks.s1.type = avro
a1.sinks.s1.channels = c1
a1.sinks.s1.hostname = flume02.example.com
a1.sinks.s1.port = 4141
a1.sinks.s1.ssl = true
a1.sinks.s1.trust-all-certs = false
a1.sinks.s1.truststore = /etc/flume-ng/ssl/flume.truststore
a1.sinks.s1.truststore-password = password
a1.sinks.s1.truststore-type = JKS
```

在 flume02.example.com 上，可以配置 AvroSource 配置，为使用证书和私钥来监听连接。密钥配置参数如下所示。

ssl

将其设为 true 以启用 SSL。启用 SSL 时，还需要对 keystore、keystore-password 和 keystore-type 参数进行配置。

keystore

将该参数设为 Java keystore 的绝对路径。

keystore-password

将该参数设为保护 keystore 的口令。

keystore-type

将其设为 JKS 或 PKCS12。

示例 10-11　Avro SSL 源的配置

```
a2.sources = r1
a2.channels = c1
a2.sources.r1.type = avro
a2.sources.r1.channels = c1
a2.sources.r1.bind = 0.0.0.0
a2.sources.r1.port = 4141
a2.sources.r1.ssl = true
a2.sources.r1.keystore = /etc/flume-ng/ssl/flume02.p12
a2.sources.r1.keystore-password = password
a2.sources.r1.keystore-type = PKCS12
```

除了保护传输中的数据，可能还需要确保数据在 Flume 传输通道中被写入磁盘时是加密的。一个方法是，在 Flume 通道中写数据的磁盘上使用支持全盘加密的第三方加密工具，

也可以通过 dm-crypt/LUKS[1] 完成。然而，全盘加密也可能是过度保护，特别是当 Flume 不是使用该日志磁盘的的唯一服务，或不是所有事件都需要被加密时。

对于那些使用场景，Flume 提供了“仅加密被文件通道使用的日志文件”的功能。当前的实现只支持 Counter 模式下无填充的 AES 加密（AES/CTR/NOPADDING），但未来有可能再添加其他的加密算法和模式。目前，Flume 只支持 **JCE keystore** 的实现（**JCEKS**）作为密钥提供程序，但也不排除添加其他密钥支持程序。这需要对 Flume 本身进行修改，因为现在并没有能支持添加密钥提供程序的可插拔接口。除了这些限制，Flume 支持密钥轮换，帮助改进其安全性。因为文件通道的日志文件的生命周期相对较短，可以视需要尽可能频繁地轮换密钥以满足需求。为了确保用以前密钥写入的日志文件仍然可读，在最新的密钥被用于写入时，也必须保留旧的密钥用于读取文件。

要设置 Flume 的文件通道磁盘加密，先生成密钥，如示例 10-12 所示。

示例 10-12　为文件通道的磁盘加密生成密钥

```
[root@flume01 ~]# mkdir keys
[root@flume01 ~]# cd keys/
[root@flume01 keys]# keytool -genseckey -alias key-0 -keyalg AES -keysize 256 \
  -validity 9000 -keystore flume.keystore -storetype jceks
Enter keystore password:
Re-enter new password:
Enter key password for <key-0>
        (RETURN if same as keystore password):
Re-enter new password:
[root@flume01 keys]#
```

示例中，将 keystore 的口令设为 `keyStorePassword`，将密钥口令设为 `keyPassword`。然而在真实环境的部署中，应当使用更强的口令。keytool 不会显示正在输入的字符，也不会显示我们熟悉的星号字符，因此需谨慎输入口令。还可以分别通过命令行的 `-storepass keyStorePassword` 和 `-keypass keyPassword` 命令提供 keystore 口令和密钥口令。通常不推荐在命令行中写入口令，因为它们一般都会被写入 shell 的历史记录文件，而该文件不能被视作安全的。接下来，将 keystore 复制到 Flume 的配置目录，如示例 10-13 所示。

示例 10-13　复制 keystore 到 Flume 的配置目录

```
[root@flume01 ~]# mkdir /etc/flume-ng/encryption
[root@flume01 ~]# cp ~/keys/flume.keystore /etc/flume-ng/encryption/
[root@flume01 ~]# cat > /etc/flume-ng/encryption/keystore.password
keyStorePassword
^D
[root@flume01 ~]# cat > /etc/flume-ng/encryption/key-0.password
keyPassword
^D
[root@flume01 ~]# chown -R root:flume /etc/flume-ng/encryption
[root@flume01 ~]# chmod 750 /etc/flume-ng/encryption
[root@flume01 ~]# chmod 640 /etc/flume-ng/encryption/*
[root@flume01 ~]#
```

注 1：为了使设置 dm-crypt/LUKS 更容易，可以使用 cryptsetup 工具。通过 cryptsetup 工具设置 dmcrypt/LUKS 的说明可以在 cryptsetup 的 FAQ 页面找到（http://bit.ly/1Hc9zCz）。

要注意，还创建了包含 keystore 口令和密钥口令的文件。屏幕显示 ^D 时，应当按住 Control 键并键入字母 D。生成这些文件有助于保护口令，因为它们不能被能够读取 Flume 配置文件的同一个用户所访问。现在配置 Flume 以启用文件通道加密，如示例 10-14 所示。

示例 10-14　加密文件通道的配置

```
a1.channels = c1
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /data/01/flume/checkpoint
a1.channels.c1.dataDirs = /data/02/flume/data,/data/03/flume/data
a1.channels.c1.encryption.cipherProvider = AESCTRNOPADDING
a1.channels.c1.encryption.activeKey = key-0
a1.channels.c1.encryption.keyProvider = JCEKSFILE
a1.channels.c1.encryption.keyProvider.keyStoreFile =
  /etc/flume-ng/encryption/flume.keystore
a1.channels.c1.encryption.keyProvider.keyStorePasswordFile =
  /etc/flume-ng/encryption/keystore.password
a1.channels.c1.encryption.keys = key-0
a1.channels.c1.encryption.keys.key-0.passwordFile =
  /etc/flume-ng/encryption/key-0.password
```



示例 10-14 至示例 10-16 展示了一些配置内容（`a1.channels.c1. encryption.keyProvider.keyStorePassword file, a1.channels.c1.encryption.keys.key-0.passwordfile`）被分成两行。这是为了改进示例的可读性，但在 Flume 配置文件中是无效的。所有配置的参数名和值都必须位于同一行。

随着时间的推移，有必要轮换至新的加密密钥，以降低旧密钥遭到破解的风险。Flume 支持配置多个解密密钥，但仅使用最新的密钥用于加密。为了保证轮换密钥前写入的旧日志文件仍然可读，旧的密钥必须保留。示例 10-15 中，可以生成一个新密钥并更新 Flume，使之成为活动密钥。

示例 10-15　为磁盘上的文件通道加密生成一个新密钥

```
[root@flume01 ~]# keytool -genseckey -alias key-1 -keyalg AES -keysize 256 \
  -validity 9000 -keystore /etc/flume-ng/encryption/flume.keystore \
  -storetype jceks
Enter keystore password:
Enter key password for <key-1>
        (RETURN if same as keystore password):
Re-enter new password:
[root@flume01 ~]# cat > /etc/flume-ng/encryption/key-1.password
key1Password
^D
[root@flume01 ~]# chmod 640 /etc/flume-ng/encryption/*
[root@flume01 ~]#
```

现在，已经向 keystore 添加了新密钥，并且创建了相关的密钥口令文件，下面可以更新 Flume 的配置以使这个新密钥生效变为活动密钥，如示例 10-16 所示。

示例 10-16　加密文件通道的新密钥配置

```
a1.channels = c1
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /data/01/flume/checkpoint
a1.channels.c1.dataDirs = /data/02/flume/data,/data/03/flume/data
a1.channels.c1.encryption.cipherProvider = AESCTRNOPADDING
a1.channels.c1.encryption.activeKey = key-1
a1.channels.c1.encryption.keyProvider = JCEKSFILE
a1.channels.c1.encryption.keyProvider.keyStoreFile =
  /etc/flume-ng/encryption/flume.keystore
a1.channels.c1.encryption.keyProvider.keyStorePasswordFile =
  /etc/flume-ng/encryption/keystore.password
a1.channels.c1.encryption.keys = key-0 key-1
a1.channels.c1.encryption.keys.key-0.passwordFile =
  /etc/flume-ng/encryption/key-0.password
a1.channels.c1.encryption.keys.key-1.passwordFile =
  /etc/flume-ng/encryption/key-1.password
```

以下是文件通道加密的配置参数。

`encryption.activeKey`
: 用于加密新数据的密钥别名。

`encryption.cipherProvider`
: 密码算法提供程序的类型。支持的类型：`AESCTRNOPADDING`

`encryption.keyProvider`
: 密钥提供程序的类型。支持的类型：`JCEKSfiLE`

`encryption.keyProvider.keyStorefile`
: keystore 文件的路径。

`encryption.keyProvider.keyStorePasswordfile`
: 含有 keystore 口令的文件的路径。

`encryption.keyProvider.keys`
: 有限长的曾用或现用的密钥别名列表。

`encryption.keyProvider.keys.<key>.passwordfile`
: 包含密钥 `key` 的口令文件的可选路径。如果忽略该参数，keystore 口令文件中的口令会被用于所有密钥。

10.2.2　Sqoop加密

与 Flume 不同，Sqoop 没有自己原生的对线上加密的支持。这并不奇怪，因为 Sqoop 依赖于标准的 JDBC 驱动和专门针对数据库优化的连接器。然而，Sqoop 可以被配置为连接到一个支持 SSL 的数据库并启用 SSL，这将启用对 JDBC 通道中数据的加密。

上述的支持并不仅限于通用的 JDBC 实现。如果用于实现 `--direct` 模式的工具支持 SSL，甚至可以在使用直接连接器的时候加密数据。

下面看看如何使用 SSL 加密 Sqoop 和 MySQL 之间的流量[1]。示例 10-17 和示例 10-18 假设 MySQL 已配置 SSL[2]。如果尚未这样设置，可以从 MySQL 连接器的下载页面（http://dev.mysql.com/downloads/connector/j/5.1.html）下载 MySQL JDBC 驱动。下载完成后，将其安装到一个 Sqoop 可用的位置，如示例 10-17 所示。

示例 10-17　为 Sqoop 安装 MySQL JDBC 驱动

```
[root@sqoop01 ~]# SQOOP_HOME=/usr/lib/sqoop
[root@sqoop01 ~]# tar -zxf mysql-connector-java-*.tar.gz
[root@sqoop01 ~]# cp mysql-connector-java-*/mysql-connector-java-*-bin.jar \
  ${SQOOP_HOME}/lib
[root@sqoop01 ~]#
```

驱动安装到位后，可以使用 Sqoop 的 `list-tables` 命令测试连接，如示例 10-18 所示。

示例 10-18　用 list-tables 测试 SSL 连接

```
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=false"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ sqoop list-tables --connect ${URI} \
  --username sqoop -P
Enter password:
cities
countries
normcities
staging_cities
visits
[alice@sqoop01 ~]$
```

设置 MySQL JDBC 驱动使用 SSL 加密的参数，包含在传递给 Sqoop 的 JDBC URI 选项中。

`verifyServerCertificate`

　该参数控制客户端是否验证 MySQL 服务器的证书。如果设为 `true`，还需要对参数 `trustCertificateKeyStoreUrl`，`trustCertificateKeyStoreType` 和 `trustCertificateKeyStorePassword` 进行设置。

`useSSL`

　该参数设为 `true` 时，客户端会尝试使用 SSL 与 server 进行会话。

`requireSSL`

　该参数设为 `true` 时，若服务器不支持 SSL，客户端会拒绝连接。

示例 10-19 中，试着通过 SSL 导入一个表。

注 1：Sqoop 示例基于 Kathleen Ting 和 Jarek Jarcec Cecho 合著的 *Apache Sqoop Cookbook*。使用的示例文件和脚本可以从 Apache Sqoop Cookbook 项目页面（http://github.com/jarcec/Apache-Sqoop-Cookbook）获取。

注 2：如果 MySQL 尚未配置 SSL，可以按照 MySQL 手册（http://dev.mysql.com/doc/refman/5.7/en/ssl-connections.html）中的说明进行配置。

示例 10-19　通过 SSL 导入一个 MySQL 表

```
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=false"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ sqoop import --connect ${URI} \
  --username sqoop -P --table cities
Enter password:
...
14/06/27 16:09:07 INFO mapreduce.ImportJobBase: Retrieved 3 records.
[alice@sqoop01 ~]$ hdfs dfs -cat cities/part-m-*
1,USA,Palo Alto
2,Czech Republic,Brno
3,USA,Sunnyvale
[alice@sqoop01 ~]$
```

可以看出，这和在 JDBC URI 中包含 SSL 参数一样简单。可以通过查看作业历史服务器页面中的作业配置确认 SSL 参数在作业执行时被使用，如图 10-2 所示。

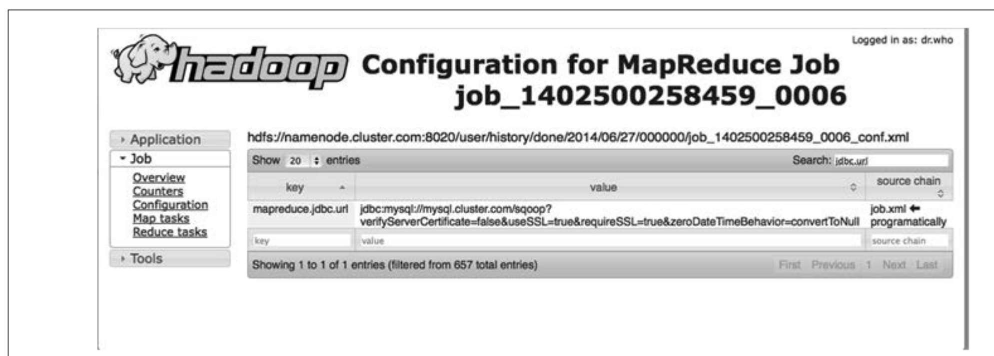


图 10-2：作业历史服务器页面，展示了 SSL JDBC 配置的使用

前例将 `verifyServerCertificate` 设为 `false`。这有助于我们测试，但实际的产品设置中，我们更愿意能够验证正在连接的服务器确实是希望连接的那一台。示例 10-20 将这个参数设为 `true`。

示例 10-20　没有 truststore 时证书验证失败

```
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=true"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ sqoop list-tables --connect ${URI} \
  --username sqoop -P
Enter password:
14/06/30 10:52:29 ERROR manager.CatalogQueryManager: Failed to list tables
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

The last packet successfully received from the server was 1,469 milliseconds ago. Th
e last packet sent successfully to the server was 1,464 milliseconds ago.
```

```
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
...
        at org.apache.sqoop.Sqoop.main(Sqoop.java:240)
Caused by: javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorExcep
tion: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderExc
eption: unable to find valid certification path to requested target
...
[alice@sqoop01 ~]$
```

毫不意外，这样行不通，因为 Java 标准证书的 truststore 不认为我们的 MySQL 服务器证书是可信任证书。诊断这些信任类型问题时，要注意的关键错误消息是 `unable to find valid certifica tion path to requested target`。这通常意味着，服务器没有得到我们受信任证书的签名。最简单的处理方法是，将 MySQL 服务器的证书导入 truststore，并且配置 MySQL JDBC 驱动在连接时使用该 truststore，如示例 10-21 所示。

示例 10-21　通过本地 truststore 列出表

```
[alice@sqoop01 ~]$ keytool \
  -import \
  -alias mysql.example.com \
  -file mysql.example.com.crt \
  -keystore sqoop-jdbc.ts
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=admin@example.com, CN=mysql.example.com, O="Cluster, Inc.",
L=San Francisco, ST=California, C=US
Issuer: EMAILADDRESS=admin@example.com, CN=mysql.example.com, O="Cluster, Inc."
, L=San Francisco, ST=California, C=US
Serial number: d7f528349bee94f3
Valid from: Fri Jun 27 13:59:05 PDT 2014 until: Sat Jun 27 13:59:05 PDT 2015
Certificate fingerprints:
         MD5: 38:9E:F4:D0:4C:14:A8:DF:06:EC:A5:59:76:D1:0C:21
         SHA1: AD:D0:CB:E2:70:C1:89:83:22:32:DE:EF:E5:2B:E5:4F:7E:49:9E:0A
         Signature algorithm name: SHA1withRSA
         Version: 1
Trust this certificate? [no]: yes
Certificate was added to keystore
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=true"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreUrl=file:sqoop-jdbc.ts
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreType=JKS"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStorePassword=password"
[alice@sqoop01 ~]$ sqoop list-tables --connect ${URI} \
  --username sqoop -P
Enter password:
cities
countries
normcities
staging_cities
visits
[alice@sqoop01 ~]$
```

本例中，首先创建一个包含 MySQL 服务器证书的 truststore，然后将 MySQL JDBC 驱动指向该 truststore。这要求在 JDBC URI 中设置一些额外的参数，如下所示。

`trustCertificateKeyStoreUrl`
: 一个指向 keystore 位置的 URL，该 keystore 用于验证 MySQL 服务器证书。

`trustCertificateKeyStoreType`
: 用于验证 MySQL 服务器证书的 keystore 类型。

`trustCertificatekeyStorePassword`
: 用于验证 MySQL 服务器证书的 Keystore 密码。

注意，我们使用了一个相对路径 `file:<URI>` 指定 truststore 的位置，它在下一个示例中很重要。列出表之后，试着在示例 10-22 中进行一次导入。

示例 10-22　通过 truststore 导入表

```
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=true"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreUrl=file:sqoop-jdbc.ts"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreType=JKS"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStorePassword=password"
[alice@sqoop01 ~]$ sqoop import \
  -files sqoop-jdbc.ts \
  --connect ${URI} \
  --username sqoop \
  -P \
  --table cities
Enter password:
...
14/06/30 10:57:13 INFO mapreduce.ImportJobBase: Retrieved 3 records.
[alice@sqoop01 ~]$
```

与上一个列出表的示例相比，变化不是很大。除了使用了一样的 URI 之外，我们还增加了一个 `-file` 命令行参数。`-file` 选项会将文件列表放入 Hadoop 的分布式缓存，该分布式缓存会把文件复制到集群中的每一个节点，并将其放置到正运行任务的工作目录下。这项功能很有用，因为这意味着我们给 `trustCertificateKeyStoreUrl` 的配置对本地机器和执行任务的所有节点都同样有效，这也正是为什么我们想用 truststore 作为启动 Sqoop 作业的工作目录。

加密支持并不仅限于普通模式，像 MySQL 的直接模式就使用了支持 SSL 的 `mysqldump` 和 `mysqlimport` 工具。示例 10-23 在直接模式中启用 SSL。

示例 10-23　使用直接模式通过 truststore 导入表

```
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=true"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreUrl=file:sqoop-jdbc.ts"
```

```
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStoreType=JKS"
[alice@sqoop01 ~]$ URI="${URI}&trustCertificateKeyStorePassword=password"
[alice@sqoop01 ~]$ sqoop import \
  -files sqoop-jdbc.ts,mysql.example.com.crt \
  --connect ${URI} \
  --username sqoop \
  -P \
  --table cities \
  --direct \
  -- \
    --ssl \
    --ssl-ca=mysql.example.com \
    --ssl-verify-server-cert
Enter password:
...
14/06/30 15:32:43 INFO mapreduce.ImportJobBase: Retrieved 3 records.
[alice@sqoop01 ~]$
```

同样，这与上一个示例也很相似。主要的区别在于，我们将 `mysql.example.com.crt` 添加到了 `-file` 选项，因而节点会拥有 `mysqldump` 工具要求的 PEM 格式的证书。还添加了 `--direct` 选项以启用直接模式。最后，增加了 `--ssl`、`--ssl-ca=mysql.example.com` 和 `--ssl-verify-server-cert` 选项。`--` 标记意味着其后所有参数应当被传给实现直接模式的工具。剩余的参数将会被 `mysqldump` 用于启用 SSL、设置 CA 证书位置以及让 `mysqldump` 验证 MySQL 服务器的证书。

10.3 导入工作流

目前，我们已经了解了数据通常如何导入 Hadoop 环境，也了解了导入管道的机密性、完整性和可用性的不同选项。然而，导入数据是一个典型的全局 ETL（抽取、转换和加载）过程，因此在全局 ETL 流的上下文环境中必须进行额外的考虑。

我们之前忽略的一个细节是，Sqoop 这类工具是从*哪里*启动的。通常情况下，你会希望限制用户能够访问的接口。正如在 3.4.1 节描述的那样，有很多种访问远程协议的途径能够得到保护，并且具体的架构依赖于用户的需求。最常见的限制边界服务（包括导入）的方法是，在*边界节点*上部署 Flume 和 Sqoop 这类服务。边界节点只不过是既能访问内部 Hadoop 集群网络、又能访问外部网络的服务器。典型的集群部署将会通过主机或网络防火墙限制对特殊主机的特殊端口的访问。数据*导入*场景中，我们能在执行并行导入时利用仍然部署着数据*拉取*机制（例如 Sqoop）的边界节点，限制向 Hadoop 集群推送数据，从而避免向外部暴露敏感的 Hadoop 服务。

“只允许远程登录边界节点”在很大程度上降低了用户物理登入 Hadoop 集群的风险，这使得能在减少潜在危险角色数量的同时，能够将精力集中在监测和安全审计工作上。生成数据流时，通常应设置专用 ETL 账户或组执行全局的工作流。对于那些需要精确审计的组织，推荐使用相互独立的用户账户发起操作，这样能使操作行为更容易追踪到个人。

除了 Flume 和 Sqoop 之外，边界节点还可能运行有代理服务或其他远程用户协议。例如，HDFS 支持 HttpFS 代理服务，该服务会向外开放一个供 HDFS 使用的读 / 写 REST 接口。

与 HDFS 一样，HttpFS 完全支持基于 Kerberos 的身份验证和 HDFS 内建的权限控制。在一个边界节点上运行 HttpFS，能够允许对 HDFS 中存储的数据进行有限的访问，甚至可以被用于某些数据的导入。

另一个常见的边界节点服务是 Oozie。Oozie 是一个规划和执行工作流的工具。由 Sqoop 作业、Hive 查询、Pig 脚本以及 MapReduce 作业组成的复杂工作流可以被组合成独立的单元，并且通过 Oozie 被可靠地规划和执行。Oozie 还提供了一个支持基于 Kerberos 身份验证的 REST 接口，并能安全地向边界节点开放。

一些用例中，有必要在把文件推送到 HDFS、HBase 或 Accumulo 之前，将其放置在一个边界节点上。创建这些本地磁盘（有时是 NFS 挂载的磁盘）目录时，应使用标准的操作系统控制手段，将对数据的访问限制为仅限授权用户。这里要再次强调，应当定义一个或多个 ETL 组，并且将对原始数据的访问范围限制为相对受信任的组。

10.4 企业架构

对于数据导入的这些讨论帮助我们明确了这样一个有用的观点：Hadoop 从来不是部署在真空之中的。Hadoop 不可避免地要与已存在的和正在演变的企业架构相结合，这意味着，不能只考虑 Hadoop 自身安全。决定如何保护集群时，必须研究已经应用于数据和系统的需求，这些需求由企业安全标准、威胁模型和特殊数据的集敏感度所驱动。需要特别指出的是，如果数据源本身就毫无安全可言，那么封锁 Hadoop 集群或集群的数据导入管道是毫无意义的。

这与企业引进数据仓库系统的情况一样。一个典型的部署中，应用程序和支持它们的事务处理系统是紧耦合的。这使得安全集成简单而直接，因为对后端数据库的访问通常被限制在生成和处理这些数据的应用程序中。一旦事务性数据重要到需要备份时，一些安全细节就被重视起来。然而，这仍然能算得上相对简单的集成，因为可以限制备份数据仅能被那些已拥有对源系统访问权的受信任管理员访问。

试图把数据从事务处理系统移动至分析型数据仓库，以便独立执行分析应用时，事情变得有趣起来。若使用传统的数据仓库系统，需要将事务数据库的安全配置与新数据仓库的特性进行对比。一旦数据被导入仓库，该机制就会工作得很好，并且可以将同样的分析工作应用于基于数据库授权的 Sentry。然而，想知道如何处理事务系统和分析平台之间的数据时，必须加以注意。

对这些传统系统的使用可以归结为保护 ETL 网格，正是这些 ETL 网格将数据加载到数据仓库。很明显，对 ETL 网格所做的考虑同样适用于 Hadoop 集群的导入管道，特别是必须考虑：在何时何地的加密措施对保护数据机密性来说才是必要的。还需要对维持数据的完整性格外加以重视。在传统的 ETL 网络中，这尤其重要，因为这种传统网格可能没有足够的存储容量在传输之后仍保留原始数据。最后，还需要对 ETL 网格的可用性加以注意，从而确保不会影响源系统或数据仓库满足用户要求的性能。这正与之前讨论的过程完全一样，即导入数据到 Hadoop 的一般过程和使用 Flume 和 Sqoop 的特定过程。

此外，需要重申，这个工作是双向的，仅在 Hadoop 导入或进行不属于源系统的查询时才

应用安全控制策略是不合理的。正如在已经开始进行已有事务性或分析性工具的安全保障设计工作后，却留着 Hadoop 门户大开一样不合理。考虑所有因素的最佳时机是在设计导入管道的时候，因为这正是 Hadoop 与其余企业架构融为一体的地方，是比对安全和威胁模型，并仔细考虑整体 Hadoop 部署的安全架构的完美时机。

10.5 小结

本章集中于数据从外部源到 Hadoop 的传输。简单讨论批量文件导入后，我们继续聚焦于使用 Flume 进行基于事件的数据导入，以及使用 Sqoop 从关系型数据库导入数据。可以发现，这些通用的数据导入机制能保护传输中的数据完整性。本章的一个关键点是，集群内部对数据的保护需要延伸至从数据源导入的全部过程。这种保护模式应当与源系统的保护等级相匹配。

我们已经了解了数据导入和集群内部数据的保护，下面学习数据保护的最后一项内容：保护数据提取过程和客户端的访问。

第11章

数据提取和客户端访问安全

Hadoop 的核心宗旨之一是，将处理工作带到数据所在之处，而不是反其道行之。因此，我们的焦点一直都是“集群内的安全如何实现”。本章将讲解保护 Hadoop 集群的最后一部分内容，也就是保护客户端的访问和数据提取的过程。虽然对 Hadoop 数据的大多数处理过程在集群内完成，但用户是通过外部工具访问数据的，并且在一些诸如企业数据仓库的用例中，还可能需要专用工具以提取海量数据。

客户端访问的最基本形式是命令行工具。正如 3.3.4 节的“边界节点”，集群中，将外部访问限制在很小的一个*边界节点*集合中是很常见的。用户使用 ssh 远程登录一个边界节点，并且使用多种命令行工具与集群交互。表 11-1 是最常用的一些命令的简要描述。

表11-1：客户端访问的常用命令行工具

命令	描述
hdfs dfs -put *<src>* *<dst>*	将本地文件复制到 HDFS
hdfs dfs -get *<src>* *<dst>*	从 HDFS 下载文件到本地系统
hdfs dfs -cat *<path>*	打印文件内容到标准输出
hdfs dfs -ls *<path>*	列出某路径下的文件和目录
hdfs dfs -mkdir *<path>*	在 HDFS 中创建一个目录
hdfs dfs -cp *<src>* *<dst>*	把 HDFS 中的文件复制到一个新的地址
hdfs dfs -mv *<src>* *<dst>*	把 HDFS 中的文件移动到一个新的地址
hdfs dfs -rm *<path>*	从 HDFS 中删除一个文件
hdfs dfs -rmdir *<path>*	从 HDFS 中删除一个目录
hdfs dfs -chgrp *<group>* *<path>*	改变一个文件或目录的组
hdfs dfs -chmod *<mode>* *<path>*	改变一个文件或目录的权限
hdfs dfs -chown *<owner>*[:*<group>*] *<path>*	改变一个文件或目录的拥有者

（续）

命令	描述
yarn jar *<jar>* [*<main-class>*] *<args>*	运行一个 JAR 文件，通常用于启动 MapReduce 作业或其他 YARN 作业
yarn application -list	列出正在运行的 YARN 应用
yarn application -kill *<app-id>*	终止一个 YARN 应用
mapred job -list	列出一个正在运行的 MapReduce 作业
mapred job -status *<job-id>*	获取一个 MapReduce 作业的状态
mapred job -kill *<job-id>*	终止一个 MapReduce 作业
hive	启动一个 Hive SQL shell（不推荐；推荐使用 beeline）
beeline	给 Hive 或 Impala 启动一个 SQL shell
impala-shell	启动一个 Impala SQL shell
hbase shell	启动一个 HBase shell
accumulo shell	启动一个 Accumulo shell
oozie job	运行、检查和终止 Oozie 作业
sqoop export	从 HDFS 导出表到数据库
sqoop import	从数据库导入表到 HDFS

11.1 Hadoop命令行接口

核心的 Hadoop 命令行工具（`hdfs`、`yarn` 和 `mapred`）只支持 Kerberos 或者使用委托令牌的方法进行身份验证。验证这些命令最简单的方式是，在命令执行前使用 `kinit` 取得 Kerberos 的票据授予票据（TGT）。[1] 如果执行 Hadoop 命令之前没有获得 TGT，你将会看到示例 11-1 所示的相似错误，特别是 `failed to find any Kerberos tgt`。

示例 11-1 在没有 Kerberos TGT 的情况下执行 Hadoop 命令

```
[alice@hadoop01 ~]$ hdfs dfs -cat movies.psv
cat: Failed on local exception: java.io.IOException: javax.security.sasl.
SaslException: GSS initiate failed [Caused by GSSException: No valid credentials
provided (Mechanism level: Failed to find any Kerberos tgt)]; Host Details : local
host is: "hadoop01.example.com/172.25.2.196"; destination host is: "hadoop02.
example.com":8020;
```

下面看看 Alice 先使用 `kinit` 取得 TGT 之后的情况（示例 11-2）。

示例 11-2 在 `kinit` 之后执行 Hadoop 命令

```
[alice@hadoop01 ~]$ kinit Password for alice@EXAMPLE.COM: [alice@hadoop01 ~]$
hdfs dfs -cat movies.psv 1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/
titleexact?Toy%20Story%20( ...
```

这次命令被成功执行，并且打印文件 `movies.psv` 中的内容。使用 Kerberos 进行身份验证的好处是，用户不需要为每个命令单独进行验证。如果在会话开始时使用 `kinit`，或将 Linux

---

注 1：回顾表 4-1 的 TGT 相关内容。

系统配置为登录时获取 Kerberos TGT，则可以运行任意数量的 Hadoop 命令，所有身份验证工作将在幕后完成。

虽然通常不这么做，但命令行工具可以使用委托令牌进行 HDFS 的身份验证。为了获得委托令牌，需要通过 Kerberos 的身份验证。HDFS 提供了一种命令行工具，以获取委托令牌并保存到一个文件。设置环境变量 HADOOP_TOKEN_fiLE_LOCATION 之后，该令牌就能被用于随后的 HDFS 命令。委托令牌的使用如示例 11-3 所示。

示例 11-3　使用委托令牌执行 Hadoop 命令

```
[alice@hadoop01 ~]$ kinit
Password for alice@EXAMPLE.COM:
[alice@hadoop01 ~]$ hdfs fetchdt --renewer alice nn.dt
14/10/21 19:19:32 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN token 2 for
 alice on 172.25.3.210:8020
Fetched token for 172.25.3.210:8020 into file:/home/alice/nn.dt
[alice@hadoop01 ~]$ kdestroy
[alice@hadoop01 ~]$ export HADOOP_TOKEN_FILE_LOCATION=nn.dt
[alice@hadoop01 ~]$ hdfs dfs -cat movies.psv
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(
...
```

委托令牌同基于 Kerberos 的身份验证是完全独立的。委托令牌一旦被分发，默认具有 24 小时的有效期，并且可以被续至最多 7 天。可以通过设置 dfs.namenode.delegation.token.renewal-interval 改变令牌的初始有效期，该参数表示了令牌需要续租前还剩下的毫秒数。可以设置 dfs.namenode.delegation.token.max-lifetime 改变令牌的最大更新周期，该配置参数的单位也是毫秒。这些令牌独立于 Kerberos 系统，所以如果 Kerberos 凭证在 KDC 中是激活的，委托令牌会在它们的指定生命周期中始终有效，没有任何管理方法能强制取消委托令牌。

Hadoop 命令行工具没有单独的授权模型，而依赖于集群的配置控制哪些内容是用户能够访问的。要复习 Hadoop 身份验证，可回顾 6.1 节 ~6.3 节。

11.2　保护应用安全

开发一个应用时，在设计方面会遇到许多选择，例如为用户接口使用什么框架，或者为后端存储使用什么系统等。应用设计中最关键的要素是，如何保护一个应用的安全性。这个问题通常由于与应用交互的大量接口而变得更加复杂。

涉及数据的授权应当在哪里进行时，通常有两种流派。一种观点认为，安全问题（特别是授权）应属于应用层。这是有道理的，因为应用程序通常与数据的控制策略有更多上下文关联。但此观点的缺点在于，这意味着每一个应用程序都必须重新实现通用的服务。随着时间的推移，数据库获得了发展，因此，应用程序将授权标签下放至数据库时，数据库可以存储和强制执行授权控制。这是给 Accumulo 添加单元级的可见性标签，和给 HBase 添加单元级的访问控制表（ACL）和可见性标签的主要动机之一。

与“授权决策在哪里进行”紧密相关的是，用户账户能深入到哪个地步？历史上，数据库一直保持着它们的私有身份目录，这通常使尝试和复制数据库目录中的用户变得复杂。

Accumulo 仍然使用自己的身份目录，因此它也具有这一缺陷。为了应对这个问题，许多应用程序的开发者采用了这样一种模式：使用数据库存储应用级账户，并由应用程序负责降级访问，以此利用数据库级的授权。

降级访问的功能要求，用户使用 Java API 访问 Accumulo 时，需要通过一系列的授权。这使得应用程序能对终端用户执行身份验证，并使用集中服务查找用户的授权。随后，应用程序会在读取数据时传递这些终端用户的授权，Accumulo 会自动将应用程序的授权信息与终端用户的授权信息交叉比对。这意味着，给终端用户提供基于他们个体级别的更细粒度的访问控制时，也可以对应用程序可访问数据的最高级别进行控制。

最后，终端用户的授权能有多深入取决于应用程序开发者的决策。随着时间的推移，应用程序和基于 Hadoop 的数据存储被集成到同一个身份目录下之后，应用程序开发者想要做出明智的设计会更加容易。安全领域并不是一成不变的，而是在持续地发展。

11.3 HBase

第 5 章中，我们了解过如何配置 HBase 以使用 Kerberos 作为身份验证机制。HBase 客户端可以通过控制台、Java API 或某个 HBase 的网关服务访问 HBase。所有的客户端访问 API 都支持 Kerberos 作为身份验证机制，且要求用户在连接前先取得 Kerberos 的 TGT。

这种客户端的访问方式依赖于具体的使用场景。对于数据库的管理访问行为，例如创建、修改或删除表，通常使用 HBase 控制台完成；使用 MapReduce 或其他数据处理框架时，通常使用 Java API；其他类型的 HBase 应用也可以直接使用 Java API，或通过网关服务访问 HBase。使用非 Java 语言访问 HBase 时，选择网关 API 的情况非常普遍。网关方式还为管理员提供了一个遏止点，以限制对 HBase 的直接访问。接下来，讨论如何安全配置 HBase 网关之前，先看看怎样通过控制台与 HBase 进行安全交互。

11.3.1 HBase shell

使用控制台时，用户通常在执行 kinit 命令之前就获得了 TGT。如果试图在没有运行 `kinit` 之前就运行控制台，将会如示例 11-4 所示。需要注意的是，栈跟踪信息末尾的错误信息 `Failed to find any Kerberos tgt`。

示例 11-4 没有 Kerberos TGT 时使用 HBase 控制台

```
[alice@hadoop01 ~]$ hbase shell
14/11/13 14:45:53 INFO Configuration.deprecation: hadoop.native.lib is deprecated.
Instead, use io.native.lib.available HBase Shell; enter 'help<RETURN>' for list of
supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0> list
TABLE
14/11/13 14:46:00 WARN ipc.RpcClient: Exception encountered while connecting
 to the server : javax.security.sasl.SaslException: GSS initiate failed [Cau
sed by GSSException: No valid credentials provided (Mechanism level: Failed
```

```
to find
any Kerberos tgt)]
14/11/13 14:46:00 FATAL ipc.RpcClient: SASL authentication failed. The most likely
cause is missing or invalid credentials. Consider 'kinit'.
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSExcepti
on: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
...

ERROR: No valid credentials provided (Mechanism level: Failed to find any
Kerberos tgt)

Here is some help for this command:
List all tables in hbase. Optional regular expression parameter could
be used to filter the output. Examples:

  hbase> list
  hbase> list 'abc.*'
  hbase> list 'ns:abc.*'
  hbase> list 'ns:.*'


hbase(main):002:0>
```

现在在示例 11-5 中再试一遍，但这次会在执行控制台前先使用 kinit 获取 TGT。

示例 11-5　执行 kinit 后使用 HBase 控制台

```
[alice@hadoop01 ~]$ kinit
Password for alice@EXAMPLE.COM:
[alice@hadoop01 ~]$ hbase shell
14/11/13 14:53:56 INFO Configuration.deprecation: hadoop.native.lib
 is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0> list
TABLE
analytics_demo
document_demo
2 row(s) in 3.1900 seconds

=> ["analytics_demo", "document_demo"]
hbase(main):002:0> whoami
alice@EXAMPLE.COM (auth:KERBEROS)
    groups: alice, hadoop-users

hbase(main):003:0>
```

HBase 控制台没有独立的授权配置，所有访问都必须通过 HBase 的授权配置进行授权，详见 6.6 节。

11.3.2 HBase REST网关

HBase 具有两种网关服务实现方式：REST 服务和 Thrift 服务。这两种实现方法都允许使用非 Java 语言访问 HBase，并且都支持身份验证、身份模拟和通过加密实现的机密性。部署哪种网关应当依赖于应用程序开发者的具体需求：通过基于 JavaScript 的 Web 应用程序直接访问网关时，通常使用 REST 接口；通过其他语言的访问则通常使用 Thrift API。现在看看怎样配置 REST 网关的身份验证功能。

第一步是为 REST 网关创建一个 Kerberos 规则，以和 HBase 的其余部分对话。这是一个服务主体，应该包括运行 REST 网关的服务器主机名——例如 `rest/rest.example.com@EXAMPLE.COM`，其中 `rest.example.com` 由运行有 REST 网关的服务器的完全限定域名代替。创建服务主体并导出含有主体密钥的文件后，需要配置 REST 服务，使其通过 Kerberos 与 HBase 集群交互。先看下面的 `hbase-site.xml` 文件：

```
<property>
  <name>hbase.rest.keytab.file</name>
  <value>/etc/hbase/conf/hbase-rest.keytab</value>
</property>
<property>
  <name>hbase.rest.kerberos.principal</name>
  <value>rest/_HOST@EXAMPLE.COM</value>
</property>
```

如果启用 HBase 授权机制，则还需要为 REST 服务使用的主体创建一个顶层 ACL。假设想授予通过 REST 网关访问的一切权限（包括管理访问权限），那么应当使用 HBase 控制台执行下列内容（详见 6.6 节）：

```
hbase(main):001:0> list grant 'rest', 'RWCA'
```

如果使用了不同的主体名，那么应当用主体的短名称代替 rest。下一步是通过 SPNEGO/Kerberos 启用 REST 客户端的验证机制。对于每一个 SPNEGO 的协议规范，需要创建一个符合 `HTTP/rest.example.com@EXAMPLE.COM` 格式的主体，其中 `rest.example.com` 用运行有 REST 网关的服务器的完全限定域名代替。对 `hbase-site.xml` 进行如下设置，以启用验证机制：

```
<property>
  <name>hbase.rest.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>hbase.rest.authentication.kerberos.principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>hbase.rest.authentication.kerberos.keytab</name>
  <value>/etc/hbase/conf/hbase-rest.keytab</value>
</property>
```

本例中，将 REST 的验证密钥表与 HBase 的验证密钥表设置为同一位置。这意味着或者需要将两种 keytab 同时导出，或者使用 `ktutil` 将两种主体的 keytab 合并为一个 keytab 文件。另外，还可以为 REST 客户端验证和 HBase 验证机制使用不同的 keytab 文件。

REST 服务通常使用 hbase.rest.kerberos.principal 同 HBase 进行身份验证，但它是代表同 REST 服务进行身份验证的用户执行操作的。为了做到这些，REST 服务必须拥有模拟其他用户的权限。我们可以使用与 5.2.4 节相同的 hadoop.proxyuser.<proxy>.groups 和 hadoop.proxyuser.<proxy>.hosts 的设置。这些设置能控制哪些用户可由代理模拟，以及它们可以从哪些主机进行身份模拟。这些设置的值是由逗号分隔的组和主机列表，* 号代表所有组 / 主机。例如，如果希望 rest 用户能模拟来自 hbase-user 组中任意主机的任意用户，需要将下列内容添加到 HBase Master 中的 hbase-site.xml 文件：

```
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.proxyuser.rest.groups</name>
  <value>hbase-users</value>
</property>
<property>
  <name>hadoop.proxyuser.rest.hosts</name>
  <value>*</value>
</property>
```

REST 服务还支持远程 REST 客户端模拟终端用户，这称为双层用户模拟，因为被 REST 客户端模拟的用户是由 REST 服务模拟的。这允许运行一个通过 REST 服务访问 HBase 的应用程序，该应用程序可以通过所有途径传递用户凭证。要想启用这种身份模拟，可将 hbase.rest.support.proxyuser 的值设为 true。可以对 hadoop.proxyuser.<app user>.groups 进行配置，控制访问 REST 服务的应用程序能模拟哪些终端用户。比如有个 whizbang 应用程序，可以模拟 whizbang-users 组内的任意用户，那么应该对 REST 服务中的 hbase-site.xml 文件进行如下设置：

```
<property>
  <name>hbase.rest.support.proxyuser</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.proxyuser.whizbang.groups</name>
  <value>whizbang-users</value>
</property>
<property>
  <name>hadoop.proxyuser.whizbang.hosts</name>
  <value>*</value>
</property>
```

图 11-1 展示了双层模拟是如何通过 HBase REST 服务工作的。终端用户 Alice 通过 LDAP 用户名和口令进行身份验证，对 Hue 证明了她的身份 (1)。然后 Hue 通过 Kerberos 使用 hue/hue.example.com@EXAMPLE.COM 主体进行身份验证，并传递了 alice 的 doAs 用户身份 (2)。最后，HBase REST 服务通过 Kerberos 使用 rest/rest.example.com@EXAMPLE.COM 主体

进行身份验证，并传递了 `alice` 的 `doAs` 用户身份 (3)。该流程在从用户到 HBase 的整条路径中有效传递了 Alice 的身份凭据。

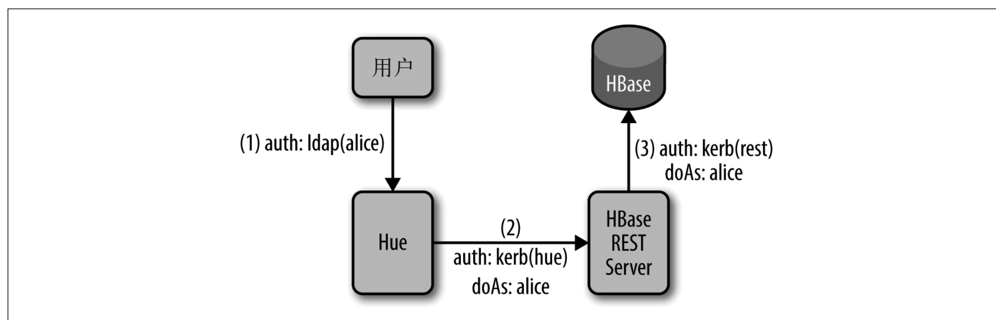


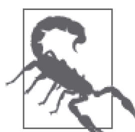
图 11-1：双层用户模拟

REST 服务可通过启用 TLS/SSL 支持客户端和 REST server 间的加密连接。要想启用 SSL，可以将 `hbase.rest.ssl.enabled` 的值设为 `true`，并将 REST 服务配置为通过私钥和证书使用 Java keystore 文件。如果 keystore 文件在 `/etc/hbase/conf/rest.example.com.jks`，并且密钥和 keystore 使用 `secret` 作为口令，那么应按照如下内容对 REST 服务的 `hbasesite.xml` 文件进行配置：

```
<property>
  <name>hbase.rest.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.rest.ssl.keystore.store</name>
  <value>/etc/hbase/conf/rest.example.com.jks</value>
</property>
<property>
  <name>hbase.rest.ssl.keystore.password</name>
  <value>secret</value>
</property>
<property>
  <name>hbase.rest.ssl.keystore.keypassword</name>
  <value>secret</value>
</property>
```

如果 REST 服务的证书没有被某个已被信任的证书签名，那么需要使用命令行工具 `keytool` 将该证书导入 Java 的 central truststore：

```
[hbase@rest ~]$ keytool -import -trustcacerts -file rest.example.com.crt \
  -keystore $JAVA_HOME/jre/lib/security/cacerts
```



上述 keytool 命令能将证书导入 Java 的 central truststore。这意味着导入的任何证书都会被任意 Java 应用程序所信任——不仅仅是 HBase，而是使用了该 JRE 的所有应用。

11.3.3 HBase Thrift网关

与 REST 网关类似，HBase Thrift 网关支持用户通过 Kerberos 进行身份验证。第一步是为 Thrift 网关创建一个 Kerberos 主体，与 HBase 其余部分交互。这是一个服务主体，并且应该包含运行 Thrift 网关的服务器主机名（如 `thrift/thrift.example.com@EXAMPLE.COM`）。创建服务主体并导出含有主体密钥的 keytab 文件后，需要配置 Thrift 服务，使其通过 Kerberos 与 HBase 集群交互。看看下面的 `hbase-site.xml` 文件：

```
<property>
  <name>hbase.thrift.keytab.file</name>
  <value>/etc/hbase/conf/hbase.keytab</value>
</property>
<property>
  <name>hbase.thrift.kerberos.principal</name>
  <value>thrift/_HOST@EXAMPLE.COM</value>
</property>
```

如果启用 HBase 的授权机制，则还需要为 Thrift 服务使用的主体创建一个顶层 ACL（访问控制表）。假设想授予通过 Thrift 网关的一切访问权限（包括管理访问权限），那么应当使用 HBase shell 执行下列内容：

```
hbase(main):001:0> list grant 'thrift', 'RWCA'
```

此时，Thrift 网关能够访问 HBase 集群，但不能进行用户身份验证。还需要将 `hbase.thrift.security.qop` 的值设为以下 3 个值之一，以启用身份验证。

`auth`
: 启用身份验证。

`auth-int`
: 启用身份验证和完整性检查。

`auth-conf`
: 启用身份验证、机密性机制（即加密）和完整性检查。

与 REST 网关类似，需要启用 Thrift 用户以模拟那些同 Thrift 网关进行身份验证的用户。同样地，将会用到 HBase Master 中 `hbase-site.xml` 文件的 `hadoop.proxyuser` 配置：

```
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.proxyuser.thrift.groups</name>
  <value>hbase-users</value>
</property>
<property>
  <name>hadoop.proxyuser.thrift.hosts</name>
  <value>*</value>
</property>
```

与 REST 网关不同的是，Thrift 网关不支持应用程序用户模拟终端用户。这意味着，如果

一个应用程序通过 Thrift 网关访问 HBase，那么其所有访问行为都会被作为 `app` 用户得到处理。即将到来的 HBase 1.0 添加了一项功能：Thrift 网关的传输方式被配置为 HTTPS 时的身份模拟。这方面的工作可以在 HBASE-12640（http://issues.apache.org/jira/browse/HBASE-12640）中找到。

图 11-2 展示了 Thrift 网关是如何进行身份模拟的。假设有一个通过 Thrift 网关访问 HBase 的 Web 应用。用户 Alice 通过 Web 应用使用 PKI（公钥基础设施）进行身份验证 (1)，然后应用程序使用 Kerberos 与 Thrift 网关进行身份验证 (2)。由于仅支持单层身份模拟，Thrift 网关使用 `thrift/thrift.example.com@EXAMPLE.COM` 主体和 `app` 的 `doAs` 与 HBase 进行身份验证 (3)。HBase 不会知道原始的终端用户是 Alice，并且向 Alice 展示结果之前，由这个 Web 应用负责申请其他授权控制。请回顾图 11-1，并比对单层和双层的用户模拟。

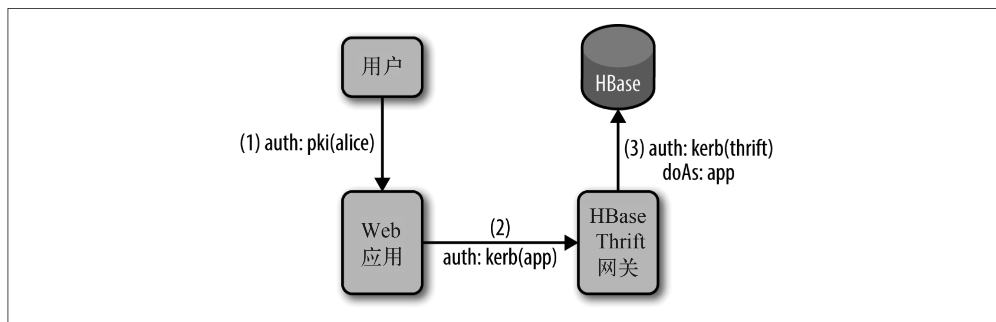


图 11-2：Thrift 网关的应用层身份模拟

11.4 Accumulo

Accumulo 客户端的访问可以通过两种机制完成：控制台和代理服务。Accumulo 控制台与 HBase 控制台类似，然而 Accumulo 的代理服务却和 HBase 的 thrift 网关服务类似。

11.4.1 Accumulo shell

与 HBase 不同的是，Accumulo 使用用户名和口令进行身份验证。支持客户端 Kerberos 验证的机制会在 Accumulo 1.7.0 版中到来，可以在 ACCUMULO-1815（http://issues.apache.org/jira/browse/ACCUMULO-2815）中对其进行跟踪。这意味着，客户端连接 Accumulo 时，必须同时提供用户名和口令。使用 Accumulo 控制台时，可以使用 `-u` 或 `--user` 命令行参数传递用户名，或者可以默认使用正在运行 shell 的 Linux 用户名。如果不传递任何参数，Accumulo 会主动要求在标准输入中输入口令。另外，还可以使用命令行、文件或者环境变量提供口令，这些方法可分别通过参数 `-p` 或 `--password` 的 `pass:<lit eral password>`、`file:<path to file with password>` 或者 `env:<environment variable with password>` 选项得到启用。还可以在运行 Accumulo 控制台之后，使用 `user` 命令改变当前用户，新用户将会被要求输入口令。在 Accumulo 控制台中传递口令的多种方法见示例 11-6。注意，提供错误口令时，控制台会打印 `Username or Password is Invalid` 消息。

示例 11-6　通过 Accumulo 控制台进行身份验证

```
[alice@hadoop01 ~]$ accumulo shell
Password: ***
2014-11-13 15:19:54,225 [shell.Shell] ERROR: org.apache.accumulo.core.client
.AccumuloSecurityException: Error BAD_CREDENTIALS for user alice - Username
or Password is Invalid
[alice@hadoop01 ~]$ accumulo shell
Password: ******

Shell - Apache Accumulo Interactive Shell
-
- version: 1.6.0
- instance name: accumulo
- instance id: 382edcfb-5078-48b4-8570-f61d92915015
-
- type 'help' for a list of available commands
-
alice@accumulo> quit
[alice@hadoop01 ~]$ accumulo shell -p pass:secret

Shell - Apache Accumulo Interactive Shell
-
- version: 1.6.0
- instance name: accumulo
- instance id: 382edcfb-5078-48b4-8570-f61d92915015
-
- type 'help' for a list of available commands
-
alice@accumulo> quit
[alice@hadoop01 ~]$ accumulo shell -p file:accumulo_pass.txt

Shell - Apache Accumulo Interactive Shell
-
- version: 1.6.0
- instance name: accumulo
- instance id: 382edcfb-5078-48b4-8570-f61d92915015
-
- type 'help' for a list of available commands
-
alice@accumulo> quit
[alice@hadoop01 ~]$ accumulo shell -p env:ACCUMULO_PASS

Shell - Apache Accumulo Interactive Shell
-
- version: 1.6.0
- instance name: accumulo
- instance id: 382edcfb-5078-48b4-8570-f61d92915015
-
- type 'help' for a list of available commands
-
alice@accumulo> user bob
Enter password for user bob: ***
bob@accumulo>
```

Accumulo 控制台没有属于自己的授权配置机制，所有访问会由 Accumulo 的授权配置机制进行授权，详见 6.6 节。

11.4.2 Accumulo代理服务

Accumulo 有一个代理服务与 HBase 的 Thrift 网关很相似，该代理服务可以被部署在任意能使用 Java 客户端 API 的服务器上。具体来说，这意味着该服务器必须能够与 Accumulo Master、ZooKeeper quorum、NameNode 和 DataNode 进行通信。对 Accumulo 代理服务的配置在 `$ACCUMULO_HOME/proxy/proxy.properties` 文件中进行，参数 `protocolFactory` 决定了服务器和客户端使用的下层 Thrift 协议。如果要改变该参数的设置，必须与客户端正在使用的协议实现相符合。如果需要支持多种 Thrift 协议，则应当部署多个代理服务。

其他必须在客户端和代理服务之间同步的设置是 `tokenClass`。代理服务不会直接对客户端进行身份验证，而会将用户提供的身份验证令牌传给 Accumulo。若需要同时支持多种身份验证令牌，也需要部署多个代理服务。

`proxy.properties` 文件的示例如下：

```
protocolFactory=org.apache.thrift.protocol.TCompactProtocol$Factory
tokenClass=org.apache.accumulo.core.client.security.tokens.PasswordToken
port=42424
instance=accumulo-instance
zookeepers=zoo-1.example.com,zoo-2.example.com,zoo-3.example.com
```

由于 Accumulo 会把身份验证令牌从访问代理服务的应用传递给 Accumulo，这相当于一个单层身份模拟，如图 11-2 所示。Accumulo 支持应用程序用户的降级访问，所以 Web 应用可以查阅 Alice 的授权，并且让 Accumulo 的授权过滤器限制 Alice 对数据的授权访问。

11.5 Oozie

从客户端访问的角度看，Oozie 是个非常重要的工具。除了作为集群工作流的执行者和规划者之外，Oozie 还可以作为给客户端的网关服务提交任意类型的作业。这允许在屏蔽从客户端对 YARN 或 MR1 的直接访问的同时，仍能允许远程的作业提交。如果选择使用该类架构，那么为 Oozie 服务启用身份验证和授权机制会非常重要。5.2.5 节第 4 小节描述过如何配置 Kerberos 身份验证机制，6.5 节详述过其授权机制。

一旦在服务器端启用了那些特性参数，那么在客户端几乎不需要额外配置也能使用。要通过 Oozie 的身份验证，仅需要在自己的工作站上缓存 Kerberos 的 TGT。这很容易实现，只要在命令行执行 Oozie 命令之前就执行 kinit 即可。若执行了一个 Oozie 命令，并且看见了 `Failed to find any Kerberos tgt` 的错误消息，这说明很可能没有运行 `kinit`：

```
[alice@edge01 ~]$ oozie jobs -oozie http://oozie01.example.com:11000/oozie
Error: AUTHENTICATION : Could not authenticate, GSSException: No valid crede
ntials provided (Mechanism level: Failed to find any Kerberos tgt)
[alice@edge01 ~]$ klist
klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc_1236000001)
[alice@edge01 ~]$ kinit
```

```
Password for alice@EXAMPLE.COM:
[alice@edge01 ~]$ oozie jobs -oozie http://oozie01.example.com:11000/oozie
No Jobs match your criteria!
```

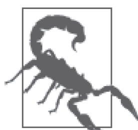
我们已经配置了 Oozie 的身份验证和授权机制，但还没有做任何操作以保证 Oozie 客户端和服务器之间通信的机密性。幸运的是，Oozie 支持使用 HTTPS 加密连接，并且提供完整性检查。要启用 HTTPS，必须获取由证书颁发机构（CA）分发给 Oozie 服务的证书。关于如何创建一个自签名证书的示例，详见 10.2.1 节。

一旦证书颁发机构分发了证书，且拥有证书和 PKCS12 格式的私钥，那么可以向 Java keystore 文件导入证书的私钥。下列示例中，给 keystore 和证书的私钥使用同样的通行码 `secret`：

```
[root@oozie01 ~]# mkdir /etc/oozie/ssl
[root@oozie01 ~]# keytool -v -importkeystore \
  -srckeystore /etc/pki/tls/private/oozie01.example.com.p12 \
  -srcstoretype PKCS12 \
  -destkeystore /etc/oozie/ssl/oozie01.example.com.keystore -deststoretype JKS \
  -deststorepass secret -srcalias oozie01.example.com -destkeypass secret
Enter source keystore password:
[Storing /etc/oozie/ssl/oozie01.example.com.keystore]
[root@oozie01 ~]# chown -R oozie:oozie /etc/oozie/ssl
[root@oozie01 ~]# chmod 400 /etc/oozie/ssl/*
[root@oozie01 ~]# chmod 700 /etc/oozie/ssl
```

接下来，在 `oozie-env.sh` 文件中设置控制 keystore 位置和口令的环境变量：

```
export OOZIE_HTTPS_KEYSTORE_FILE=/etc/oozie/ssl/oozie01.example.com.keystore
export OOZIE_HTTPS_KEYSTORE_PASS=secret
```



此处的 keystore 口令会对任何能够在运行 Oozie 的服务器上执行进程的人可见。必须对 keystore 文件采用强的权限保护措施，以防止用户读取或修改 keystore 文件。

配置 Oozie 使用 HTTPS 之前，需要确保 Oozie 服务没有运行。要配置 Oozie 使用 HTTPS，可以运行以下命令：

```
[oozie@oozie01 ~]$ oozie-setup.sh prepare-war -secure
```

现在，如果启用该服务，将会在端口 11443 上使用 HTTPS 协议。可以通过设置 `oozie-env.sh` 文件中的环境变量 `OOZIE_HTTPS_PORT` 改变端口。

在访问 Oozie 的客户端机器上，可以简单地通过命令行将 Oozie URL 改为 https://oozie01.example.com:11443/oozie，如：

```
[alice@edge01 ~]$ oozie jobs -oozie https://oozie01.example.com:11443/oozie
No Jobs match your criteria!
```

如果没有获得期望的输出，而得到了如下 `SSLHandshakeException` 错误消息：

```
[alice@edge01 ~]$ oozie jobs -oozie https://oozie01.example.com:11443/oozie
Error: IO_ERROR : javax.net.ssl.SSLHandshakeException: sun.security.validator.
```

```
ValidatorException: PKIX path building failed: sun.security.provider.certpath.
SunCertPathBuilderException: unable to find valid certification path torequested
target
```

这说明，Oozie 服务使用的证书没有被信任的证书颁发机构签名。这可能发生在使用自签名证书或某个内部 CA 没有被根 CA 签名的情况下。假设 `example-ca.crt` 文件中有一个 `EXAMPLE.COM` 域的证书颁发机构，由于要将该证书导入 Java 的 central truststore，所以该证书会被所有运行于该服务器上的 Java 应用所信任，而不仅仅是 Oozie。可以通过以下命令将其导入 Java 的 central truststore：

```
[root@edge01 ~]# keytool -import -alias EXAMPLE.COM -file example-ca.crt \
  -keystore ${JAVA_HOME}/jre/lib/security/cacerts
Enter keystore password:
Owner: CN=Certificate Authority, O=EXAMPLE.COM
Issuer: CN=Certificate Authority, O=EXAMPLE.COM
Serial number: 1
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

Java cacerts 文件的默认口令是 `changeit`。如果 Oozie 被配置为高可用性（HA），那么需要将负载均衡器配置进行 TLS（传输层安全）直连。这会允许客户端看见 Oozie 服务器的证书，且不需要负载均衡器有自己的证书。进行 TLS 直连操作时，应当使用通配证书或者包含负载均衡器全限定域名作为有效名称的证书。

11.6 Sqoop

第 10 章中，我们讨论了如何保护机密性、完整性和数据导入管道的可用性，这些原则也同样适用于保护数据提取管道。Sqoop 中，机密性不是由 Sqoop 本身提供的，但可以由 Sqoop 使用的、与 RDBMS 服务交互的驱动提供。10.2.2 节展示了应该怎样配置 MySQL 驱动，以使用 SSL 对 MySQL 服务器和 Sqoop 执行的任务之间的通信进行加密。可以使用相同的参数配置加密那些正在导出的数据，如示例 11-7 所示。

示例 11-7　用 SSL 导出 MySQL 表

```
[alice@sqoop01 ~]$ hdfs dfs -cat cities/*
1,USA,Palo Alto
2,Czech Republic,Brno
3,USA,Sunnyvale
[alice@sqoop01 ~]$ URI="jdbc:mysql://mysql01.example.com/sqoop"
[alice@sqoop01 ~]$ URI="${URI}?verifyServerCertificate=false"
[alice@sqoop01 ~]$ URI="${URI}&useSSL=true"
[alice@sqoop01 ~]$ URI="${URI}&requireSSL=true"
[alice@sqoop01 ~]$ sqoop export --connect ${URI} \
  --username sqoop -P --table cities \
  --export-dir cities
Enter password:
...
14/06/28 17:27:22 INFO mapreduce.ExportJobBase: Exported 3 records.
[alice@sqoop01 ~]$
```

11.7 SQL访问

正如第 1 章所述，使用 SQL 访问 Hadoop 数据有两种流行的方法：Hive 和 Impala。这二者都支持 Kerberos 和基于 LDAP 的用户名 / 口令身份验证。用户通常不直接与 Hive 或 Impala 进行交互，而依赖 SQL shell 或 JDBC 驱动。

本章的剩余内容包括配置 Impala 和 Hive 支持的身份验证协议，以及作为客户端如何传递验证信息。Impala 和 Hive 的授权机制由 Sentry 提供，该部分已经在第 7 章讲过。

11.7.1 Impala

Impala 能被配置为仅使用 Kerberos、仅使用 LDAP 或同时使用 LDAP 和 Kerberos 进行身份验证。Impala 运行于启用 Kerberos 的 Hadoop 集群中时，必须被配置为使用 Kerberos，这样 Impala 才能与 HDFS 和 YARN 安全地进行通信。

1. 使用带有Kerberos验证机制的Impala

与 Hadoop 的通信启用了 Kerberos 时，基于 Kerberos 的客户端身份验证机制会自动启用。Impala 使用命令行工具进行配置，应当对 `impalad`、`statestored` 和 `catalogd` 守护进程的 `--principal` 和 `--keytab_file` 参数进行配置。`--principal` 应被配置为 Impala 进行身份验证的 Kerberos 主体，通常格式是 `impala/<fully qualified domain name>@<realm>`，其中 `<fully qualified domain name>` 是运行着 `impalad` 的主机名，`<realm>` 是 Kerberos 域。主体的第一个组件 `impala` 必须与启动 Impala 进程的用户名相匹配。`--keytab_file` 参数必须指向一个包含前述主体和运行有 `impalad` 服务器的 HTTP 主体的 keytab 文件。可以借助 `ktutil` 命令，用来自两个独立 keytab 的文件创建一个 keytab 文件，如示例 11-8 所示。

示例 11-8　融合 Impala 和 HTTP 的 keytab

```
[impala@impala01 ~]$ ktutil
ktutil: rkt impala.keytab
ktutil: rkt http.keytab
ktutil: wkt impala-http.keytab
ktutil: quit
```

为了使配置更容易，可以对 `impalad` 进程使用的命令行参数进行设置，设置 `/etc/default/impala` 文件中的 `IMPALA_SERVER_ARGS`、`IMPALA_STATE_STORE_ARGS` 和 `IMPALA_CATALOG_ARGS` 变量即可。

示例 11-9　为 Impala 配置 Kerberos 身份验证

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --principal=impala/impala01.example.com@EXAMPLE.COM \
  --keytab_file=/etc/impala/conf/impala-http.keytab"

IMPALA_STATE_STORE_ARGS="${IMPALA_STATE_STORE_ARGS} \
  --principal=impala/impala01.example.com@EXAMPLE.COM \
  --keytab_file=/etc/impala/conf/impala-http.keytab"

IMPALA_CATALOG_ARGS="${IMPALA_CATALOG_ARGS} \
  --principal=impala/impala01.example.com@EXAMPLE.COM \
  --keytab_file=/etc/impala/conf/impala-http.keytab"
```

如果用户在负载均衡之后访问 Impala，那么配置需要一些小小的改动。创建融合的 keytab 文件时，还需要加入代理服务主体的 keytab，并且添加 `--be_principal` 参数。`--be_principal` 参数是 Impala 与诸如 HDFS 等后端服务进行通信的主体，其值应与之前设置的 `--principal` 参数一样，且 `--principal` 参数应被改为负载均衡的主体。若负载均衡器在 `impala-proxy.example.com` 上，那么应当如示例 11-10 所示设置 `IMPALA_SERVER_ARGS`。

示例 11-10　配置负载均衡器之后的 Impala 的 Kerberos 验证机制

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --principal=impala/impala-proxy.example.com@EXAMPLE.COM \
  --be_principal=impala/impala01.example.com@EXAMPLE.COM \
  --keytab_file=/etc/impala/conf/impala-http.keytab"
```

Impala 支持用户通过 Llama 项目使用 YARN，以进行资源管理。Llama 能协调 YARN 和 Impala 这种低时延执行引擎之间的资源管理。Llama 有两个组件：一个长期运行的应用控制程序，一个节点管理插件。应用控制程序处理 Impala 的保留资源，节点管理插件与本地 Impala 守护程序合作调节本地节点的可用资源。

Impala 启用 Kerberos 时，必须对 `llama-site.xml` 文件中的下列参数进行设置，以配置 Llama 的 Kerberos。

`lama.am.server.thrift.security`
: 将其设为 `true`，以便为应用控制程序启用 Thrift SASL/ 基于 Kerberos 的安全策略。

`llama.am.server.thrift.security.QOP`
: 启用安全策略时，用其设置保护级别。有效参数为：仅进行身份验证设为 `auth`，进行身份验证和完整性检查设为 `auth-int`，进行身份验证、完整性检查和机密性保障（加密）则设为 `auth-conf`。

`llama.am.server.thrift.kerberos.server.principal.name`
: Llama 应用服务器的完全限定主体名。该配置必须同时包含运行有 Llama 应用控制程序的服务器的短名称和完全限定主机名。

`llama.am.server.thrift.kerberos.notification.principal.name`
: 用于客户端通知的短名称。该短名称结合了注册期间 impalad 进程提供的客户端主机名。可以通过配置 `IMPALA_SERVER_ARGS` 变量中的 `--hostname` 参数覆盖该主机名。

示例 11-11 展示了 `llama-site.xml` 文件配置启用 Kerberos 的片段。

示例 11-11　配置 Llama 应用控制程序的 Kerberos 验证机制

```
<property>
  <name>llama.am.server.thrift.security</name>
  <value>true</value>
</property>
<property>
  <name>llama.am.server.thrift.kerberos.keytab.file</name>
  <value>/etc/llama/conf/llama.keytab</value>
</property>
<property>
  <name>llama.am.server.thrift.kerberos.server.principal.name</name>
```

```
  <value>llama/llama.example.com@EXAMPLE.COM</value>
</property>
<property>
  <name>llama.am.server.thrift.kerberos.notification.principal.name</name>
  <value>impala</value>
</property>
```

一旦 Impala 被配置为使用 Kerberos 验证机制，客户端就可以通过缓存的 Kerberos TGT 进行身份验证（如在执行控制台之前运行 `kinit`）。示例 11-12 展示了 Alice 获取其 Kerberos TGT 并使用 Impala 控制台进行 Kerberos 身份验证的过程。

示例 11-12　使用 Impala 控制台进行 Kerberos 身份验证

```
[alice@hadoop01 ~]$ kinit
Password for alice@EXAMPLE.COM:
[alice@hadoop01 ~]$ impala-shell -i impala-proxy
Starting Impala Shell without Kerberos authentication
Error connecting: TTransportException, TSocket read 0 bytes
Kerberos ticket found in the credentials cache, retrying the connection with a s
ecure transport.
Connected to impala-proxy:21000
Server version: impalad version 2.0.0 RELEASE (build ecf30af0b4d6e56ea80297df218
9367ada6b7da7)
Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell v2.0.0 (ecf30af) built on Sat Oct 11 13:56:06
 PDT 2014)
[impala-proxy:21000] > show tables;
Query: show tables
+-----------+
| name      |
+-----------+
| sample_07 |
| sample_08 |
+-----------+
Fetched 2 row(s) in 0.18s
[impala-proxy:21000] >
```

通过 JDBC 驱动进行 Kerberos 身份验证时，首先需要获取一个 Kerberos TGT，然后将 Impala 主体的名称包含在连接字符串中。示例 11-13 展示了如何为 Impala 的 Kerberos 验证设置 JDBC 连接字符串。

示例 11-13　用于 Kerberos 身份验证的 JDBC 连接字符串

```
//首先定义一个基本的JDBC URL字符串
String url = "jdbc:hive2://impala-proxy.example.com:21050/default";

//添加Impala Kerberos主体名,包括FQDN和域
url = url + ";principal=impala/impala-proxy.example.com@EXAMPLE.COM";

// 使用URL创建连接
Connection con = DriverManager.getConnection(url);
```

2. 使用带有LDAP/ Active Directory验证的Impala

Impala 还可以被配置为使用 LDAP 目录进行身份验证，例如 Active Directory。使用 LDAP 验证的优势在于，其客户端不需要在连接到 Impala 之前先获取它们的 Kerberos 证书。这个优势对于那些可能在本地不支持基于 Kerberos 验证的商业智能软件来说，是很有帮助的。虽然 LDAP 身份验证的配置独立于 Kerberos，但二者通常会被一起配置，因为 Hadoop 没有启用身份验证机制时，只单独给 Impala 启动身份验证机制也没有多大意义。

设置 impalad 守护进程中的 --enable_ldap_auth 和 --ldap_uri parameters 参数，启用 LDAP 身份验证机制。配置 LDAP 时，可以根据使用的 LDAP 的提供者选择性地设置绑定参数。若使用 Active Directory，则通常不需要进行额外的配置，但需要明确设置域名，从而使与 AD（Active Directory）绑定的用户名以 user@<domain name> 的形式传递。该域名通过参数 --ldap_domain 进行设置。对于 OpenLDAP 或 freeIPA，可以配置一个基准标识域名，从而使与 LDAP 绑定的用户名以 uid=user,<base dn> 的形式传递。该设置通过参数 --ldap_baseDN 启用。若 LDAP 提供者不使用 uid=user 指定用户名中的唯一性名称，则可以提供一个会被当成唯一性名称的格式。通过用首选绑定的用户名取代所有的 #UID 实例得到实现。该设置通过配置参数 --ldap_bind_pattern 启用。

不考虑 LDAP 提供者，强烈推荐使用 TLS 加密 Impala 和 LDAP 服务器之间的连接，可以通过一个 ldaps://URL 或启用 StartTLS 完成。可以将参数 --ldap_tls 设为 true 以启用 StartTLS。对于任一种模式，必须配置证书颁发机构（CA）的证书，以使 Impala 能够信任 LDAP 服务器的证书。可以通过设置参数 --ldap_ca_certificate 对 CA 证书的位置进行配置。

示例 11-14~ 示例 11-16 分别使用 Active Directory、OpenLDAP 和自定义 LDAP 提供者进行配置。

示例 11-14　配置使用 Active Directory 的 Impala

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --enable_ldap_auth=true \
  --ldap_uri=ldaps://ad.example.com \
  --ldap_ca_certificate=/etc/impala/pki/ca.crt \
  --ldap_domain=example.com"
```

示例 11-15　配置使用 OpenLDAP 的 Impala

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --enable_ldap_auth=true \
  --ldap_uri=ldaps://ldap.example.com \
  --ldap_ca_certificate=/etc/impala/pki/ca.crt \
  --ldap_baseDN=ou=People,dc=example,dc=com"
```

示例 11-16　配置使用其他 LDAP 的 Impala

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --enable_ldap_auth=true \
  --ldap_uri=ldaps://ldap.example.com \
  --ldap_ca_certificate=/etc/impala/pki/ca.crt \
  --ldap_bind_pattern=user=#UID,ou=users,dc=example,dc=com"
```

要使用 LDAP 验证机制连接 Impala，将命令行的 `-l` 选项配置为 `impala-shell`，连接建立完成前会要求键入用户口令。如果想用与当前 Linux 用户不同的另一个用户身份进行连接，可以使用 `-u` 选项更改用户名。示例 11-17 展示了如何通过 Impala 控制台使用 LDAP 进行身份验证。

示例 11-17　使用 LDAP/Active Directory 身份验证的 Impala 控制台

```
[alice@hadoop01 ~]$ impala-shell -i impala-proxy -l
Starting Impala Shell using LDAP-based authentication
LDAP password for alice:
Connected to impala-proxy:21000
Server version: impalad version 2.0.0 RELEASE (build ecf30af0b4d6e56ea80297d
f2189367ada6b7da7)
Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell v2.0.0 (ecf30af) built on Sat Oct 11 13:5
6:06 PDT 2014)
[impala-proxy:21000] > show tables;
Query: show tables
+-----------+
| name      |
+-----------+
| sample_07 |
| sample_08 |
+-----------+
Fetched 2 row(s) in 0.16s
[impala-proxy:21000] >
```

如果使用 JDBC 驱动连接 Impala，则需要在建立连接时将用户名和口令传递给 DriverManager。示例 11-18 展示了如何使用 JDBC 驱动连接使用 LDAP 验证的 Impala。

示例 11-18　用于 LDAP/Active Directory 身份验证的 KDBC 连接串

```
// 定义一个基本的JDBC URL字符串
String url = "jdbc:hive2://impala-proxy.example.com:21050/default";

// 使用URL、用户名和密码创建连接
Connection con = DriverManager.getConnection(url, "alice", "secret");
```

3. 为Impala启用SSL传输加密

之前介绍的方法涵盖了从客户端进行身份验证的不同途径。为客户端和 Impala 之间的数据传输建立一个受保护的通道也很重要，Impala 处理的数据较为敏感时，这更重要——例如数据需要被静态加密时。出于这些目的，Impala 支持 SSL 传输加密。示例 11-19 展示了必需的启动标识。

示例 11-19　对 Impala 进行 SSL 配置

```
IMPALA_SERVER_ARGS="${IMPALA_SERVER_ARGS} \
  --ssl_client_ca_certificate=/etc/impala/ca.cer \
  --ssl_private_key=/etc/impala/impala.key \
  --ssl_server_certificate=/etc/impala/impala.cer
```

ssl_private_key、ssl_server_certificate 和 ssl_client_ca_certificate 的路径对于 Impala 用户来说都必须是可读的，并且证书必须是 PEM 格式。推荐将私钥的权限限制为 400。

Impala 配置了 SSL 时，客户端还必须知道如何正确连接服务端。--ssl 选项能够让 Impala-shell 为连接而启用 SSL，--ca_cert 参数能指定用于验证连接的 Impala 守护程序提供证书的 CA 链（以 PEM 的格式）。示例 11-20 展示了同时使用 Kerberos 验证机制和 SSL 传输加密的情况。

示例 11-20　启用 SSL 和 Kerberos 的 Impala 控制台

```
alice@hadoop01 ~]$ impala-shell -i impala-proxy -k --ssl --ca_cert /etc/impala/ca.pem
Starting Impala Shell using Kerberos authentication
SSL is enabled
Connected to impala-proxy:21000
Server version: impalad version 2.0.0 RELEASE (build ecf30af0b4d6e56ea80297d
f2189367ada6b7da7)
Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell v2.0.0 (ecf30af) built on Sat Oct 11 13:5
6:06 PDT 2014)
[impala-proxy:21000] > show tables;
Query: show tables
+-----------+
| name      |
+-----------+
| sample_07 |
| sample_08 |
+-----------+
Fetched 2 row(s) in 0.16s
[impala-proxy:21000] >
```

11.7.2　Hive

陈旧过时的 Hive 命令行工具 hive 并不能支持直接的 Hive 身份验证或授权，而是要么直接访问 HDFS 上的数据，要么运行一个 MapReduce 作业执行查询。这意味着它与我们之前描述的 Hadoop 命令遵循着同样的规则，只支持 Kerberos 和委托令牌。hive 命令基本已被弃用，用户应当使用 beeline。

使用 beeline 或 JDBC 驱动时，用户要连接到负责解析查询和执行的 HiveServer2 服务进程。HiveServer2 支持 Kerberos、LDAP 和自定义身份验证插件。由于一次只能配置一个认证提供者（authentication provider），所以管理员配置 HiveServer2 服务进程时，需要选择其首选的身份验证机制。解决这种限制的一个方法是，运行多个共享同一 Hive 元数据的 HiveServer2 守护进程。这要求终端用户能够根据其身份验证需求连接到正确的 HiveServer2。HiveServer2 的身份验证机制可以在 hive-site.xml 文件中配置。HiveServer2 身份验证配置属性的描述见表 11-2。

表11-2： 配置HiveServer2身份验证属性

属性	描述
hive.server2.authentication	客户认证类型。有效值包括：NONE、LDAP、KERBEROS、CUSTOM
hive.server2.authentication.kerberos.principal	HiveServer2 服务进程的 Kerberos 主体
hive.server2.authentication.kerberos.keytab	用于 KDC 身份验证的 keytab
hive.server2.thrift.sasl.qop	Kerberos 连接使用的 SASL 质量保护；有效值包括：auth 表示仅身份验证；auth-int 表示身份验证和完整性验证；auth-conf 表示身份验证、完整性和机密性（加密）
hive.server2.use.SSL	设置为 true，在客户端与 HiveServer2 服务之间启用 TLS
hive.server2.keystore.path	包含 TLS 要使用的私钥的 Java keystore 文件路径
hive.server2.keystore.password	Java keystore 文件的密码
hive.server2.authentication.ldap.url	LDAP/Active Directory 服务器的 URL；仅当 hive.server2.authentication 设为 LDAP 时使用
hive.server2.authentication.ldap.Domain	进行身份验证的 Active Directory 域；仅当 hive.server2.authentication.ldap.url 指向 AD 服务器时使用
hive.server2.authentication.ldap.baseDN	hive.server2.authentication.ldap.url 指向 OpenLDAP 服务器时使用的基准标识名
hive.server2.custom.authentication.class	实现 org.apache.hive.service.auth.PasswdAuthenticationProviderinterface 的类名，hive.server2.authenticationis 设为 CUSTOM 时使用

1. 使用HiveServer2的Kerberos身份验证

配置 HiveServer2 的 Kerberos 身份验证与 5.2.5 节描述的配置其他核心 Hadoop 服务的模式相同。也就是说，需要将认证类型设为 Kerberos，并且设置 Kerberos 主体和 keytab。设置 Kerberos 主体时，可以使用 _HOST 占位符，它会被自动替换为运行 HiveServer2 服务器的完全限定域名。示例 11-21 展示了一个启用 Kerberos 身份验证的 hive-site.xml 示例片段。

示例 11-21 配置 HiveServer2 的 Kerberos 身份验证

```
<property>
  <name>hive.server2.authentication</name>
   <value>KERBEROS</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.principal</name>
  <value>hive/_HOST@EXAMPLE.COM</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.keytab</name>
  <value>/etc/hive/conf/hive.keytab</value>
</property>
```

要想连接到启用了 Kerberos 的 HiveServer2 服务进程的 JDBC 客户端，需要拥有一个有效的 Kerberos TGT，并将 HiveServer2 服务进程的主体添加到连接字符串。如示例 11-22 所示，创建一个用于 Kerberos 身份验证的连接字符串。

示例 11-22　用于 Kerberos 身份验证的 JDBC 连接字符串

```
//首先定义一个基本JDBC URL字符串
String url = "jdbc:hive2://hive.example.com:10000/default";

// 添加Hive Kerberos主体名,包括FQDN和域
url = url + ";principal=hive/hive.example.com@EXAMPLE.COM";

// 根据URL创建连接
Connection con = DriverManager.getConnection(url);
```

Beeline 控制台使用 Hive JDBC 驱动连接 HiveServer2。需要使用 kinit 获取 Kerberos TGT，然后使用 JDBC 连接字符串在 Beeline 中进行 Kerberos 身份验证，如示例 11-23 所示。即使在使用 Kerberos 进行身份验证，Beeline 依然会提示输入用户名和密码。可以将这些内容留空，直接敲回车。

示例 11-23　用于 Kerberos 身份验证的 Beeline 连接字符串

```
[alice@hadoop01 ~]$ kinit
Password for alice@EXAMPLE.COM:
[alice@hadoop01 ~]$ beeline
Beeline version 0.13.1 by Apache Hive
beeline> !connect jdbc:hive2://hive.example.com:10000/default;principal=hive/hiv
e.example.com@EXAMPLE.COM
scan complete in 2ms
Connecting to jdbc:hive2://hive.example.com:10000/default;principal=hive/hive.ex
ample.com@EXAMPLE.COM
Enter username for jdbc:hive2://hive.example.com:10000/default;principal=hive/hi
ve.example.com@EXAMPLE.COM:
Enter password for jdbc:hive2://hive.example.com:10000/default;principal=hive/hi
ve.example.com@EXAMPLE.COM:
Connected to: Apache Hive (version 0.13.1)
Driver: Hive JDBC (version 0.13.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hive.example.com> show tables;
+------------+--+
|  tab_name  |
+------------+--+
| sample_07  |
| sample_08  |
+------------+--+
2 rows selected (0.261 seconds)
0: jdbc:hive2://hive.example.com>
```

2. 使用HiveServer2的LDAP/ Active Directory身份验证

HiveServer2 也支持基于 LDAP 的用户名 / 口令身份验证。要使用基于 LDAP 的身份验证，需要将认证类型设为 `LDAP`，配置 LDAP URL，并且之后设置一个用于绑定的域名或基准标识名（base distinguished name）。绑定 Active Directory 服务器时使用域名，而绑定

OpenLDAP 或 freeIPA 等其他 LDAP 提供者时，使用基准标识名。

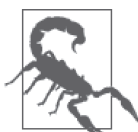


默认情况下，客户端和 HiveServer2 的连接是没有加密的。这意味着，使用 LDAP 或自定义认证提供者时，用户名和口令都有可能被第三方拦截。使用非 Kerberos 认证提供者时，强烈建议使用 TLS 开启 HiveServer2 的线上加密，详情参见后文。

无论是哪种 LDAP 提供者，我们都强烈建议不要直接使用 LDAP，而使用 LDAPS（LDAP over SSL），这能够保证 HiveServer2 和 LDAP 服务器之间的通信是加密的。要使用 LDAPS，需要确保 LDAP 服务器证书或者 CA 签名证书加载到 Java 信任库（truststore）。可以是位于 `$JAVA_HOME/jre/lib/security/cacerts` 的整个系统的 Java 信任库，也可以是用于 Hive 的特定信任库。如果使用特定的信任库，需要设置 `javax.net.ssl.trustStore` 和 `javax.net.ssl.trustStorePassword` 系统属性。这可以通过设置 `hive-env.sh` 文件中的 `HADOOP_OPTS` 变量实现，如示例 11-24 所示。

示例 11-24　为 Hive 设置 LDAPS 信任库

```
HADOOP_OPTS="-Djavax.net.ssl.trustStore=/etc/pki/java/hive.truststore"
HADOOP_OPTS="${HADOOP_OPTS} -Djavax.net.ssl.trustStorePassword=secret"
```



此处使用的信任库密码对能够在运行 HiveServer2 的服务器上枚举进程的任意用户都是可见的。必须用强制权限保护信任库文件本身，以防其他用户修改信任库。

如果要配置 HiveServer2，以对 Active Directory 服务器进行身份验证，那么除了常规的 LDAP 设置之外，还需要将 `hive-site.xml` 中的 `hive.server2.authentication.ldap.Domain` 设置为 AD 域名，如示例 11-25 所示。

示例 11-25　配置 HiveServer2 中的 Active Directory 身份验证

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>ldaps://ad.example.com</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.Domain</name>
  <value>example.com</value>
</property>
```

如果使用的是其他 LDAP 提供者，如 OpenLDAP、freeIPA 等，那么需要设置 `hive.server2.authentication.ldap.baseDN` 属性，而不是设置域名。基准 DN 取决于环境，但通常 OpenLDAP 安装的默认值是 `ou=People,dc=example,dc=com`，其中 `dc=example,dc=com` 要替换为 LDAP 服务器域组件，这通常是 LDAP 服务器的域名。对于 freeIPA，默认的基

本 DN 是 `cn=users,cn=accounts,dc=example,dc=com`，同样，也要根据环境替换其中的域组件。示例 11-26 提供了完整配置。

示例 11-26　配置 HiveServer2 中的 LDAP 身份验证

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>ldaps://ldap.example.com</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.baseDN</name>
  <value>ou=People,dc=example,dc=com</value>
</property>
```



某些版本的 Hive（尤其是 Hive 0.13.0 和 0.13.1）存在一个 bug，即身份验证类型被设为除 `KERBEROS` 之外的其他值时，它们就不会使用 Kerberos 身份验证与 Hadoop 通信。使用这些版本的 Hive 时，应当只使用 Kerberos 进行身份验证。

配置 LDAP/Active Directory 身份验证后，连接 HiveServer2 就很容易了。只需在建立连接之后，将用户名和口令传递给 DriverManager 即可，如示例 11-27 所示。

示例 11-27　LDAP/Active Directory 身份验证的 JDBC 连接字符串

```
// 使用基本JDBC URL字符串
String url = "jdbc:hive2://hive.example.com:10000/default";

// 根据URL创建提交用户名和密码的连接
Connection con = DriverManager.getConnection(url, "alice", "secret");
```

连接 Beeline 的方法基本相同。这次要在 `!connect` 命令后、出现提示时，输入用户名和口令，如示例 11-28 所示。

示例 11-28　LDAP/Active Directory 身份验证的 Beeline 连接字符串

```
[alice@hadoop01 ~]$ beeline
Beeline version 0.13.1 by Apache Hive
beeline> !connect jdbc:hive2://hive.example.com:10000/default
scan complete in 2ms
Connecting to jdbc:hive2://hive.example.com:10000/default
Enter username for jdbc:hive2://hive.example.com:10000/default: alice
Enter password for jdbc:hive2://hive.example.com:10000/default: ******
Connected to: Apache Hive (version 0.13.1)
Driver: Hive JDBC (version 0.13.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hive.example.com> show tables;
+------------+--+
|  tab_name  |
+------------+--+
```

```
| sample_07  |
| sample_08  |
+------------+--+
2 rows selected (0.261 seconds)
0: jdbc:hive2://hive.example.com>
```

3. 使用HiveServer2的可插入身份验证

Hive 具有实现新认证提供者的可插入接口，Hive 将这种身份验证模式称为 `CUSTOM`，它需要一个实现了 `org.apache.hive.service.auth.PasswdAuthenticationProvider` 接口的 Java 类。该接口定义了一个 `authenticate(String user, String password)` 方法，应当实现该方法以验证用户提交的用户名和密码。顾名思义，该可插入接口只支持使用用户名和密码进行身份验证的认证提供者。要配置这种身份验证模式，需要将认证类型设为 `CUSTOM`，并配置好身份验证类，还需要将包含身份验证类的 JAR 文件添加到 Hive 的 classpath。最简单的方法是，将 JAR 文件路径添加到 `hive-site.xml` 中的 `hive.aux.jars.path` 配置项，该配置项接受的输入为逗号分隔的 JAR 文件全路径（示例 11-29）。

示例 11-29　配置 HiveServer2 中的可插入身份验证

```
<property>
  <name>hive.server2.authentication</name>
  <value>CUSTOM</value>
</property>
<property>
  <name>hive.server2.custom.authentication.class</name>
  <value>com.example.my.whizbang.AuthenticationProvider</value>
</property>
<property>
  <name>hive.aux.jars.path</name>
  <value>file:///opt/hive-plugins/whizbang-1.0.jar</value>
</property>
```

自定义身份验证的连接设置与基于 LDAP/Active Directory 的身份验证一样（示例 11-18）。

4. HiveServer2线上加密

Hive JDBC 驱动支持两种启用线上加密的方法，使用哪种方法取决于使用的认证方法和 Hive 版本。使用 Kerberos 认证时，Hive JDBC 驱动使用 SASL 执行 Kerberos 认证。SASL 在基于一个名为*保护质量*（**quality of protection**，**QOP**）的配置项进行身份验证时，支持完整性验证和加密。要启用加密，需要将 SASL QOP 设置为 `auth-conf`，它是保密性身份验证（authentication with confidentiality）的简称。示例 11-30 展示了如何配置 HiveServer2 以使用 SASL 进行加密。

示例 11-30　配置 HiveServer2 使用 SASL 加密

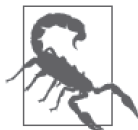
```
<property>
  <name>hive.server2.thrift.sasl.qop</name>
  <value>auth-conf</value>
</property>
```

服务端启用 SASL QOP 后，需要确保客户端也将其设置成同样的值。可以将 `sasl.qop=auth-conf` 选项添加到 JDBC URL 以实现。示例 11-31 展示了如何在 Beeline 中使用 SASL 加密。

示例 11-31　使用 SASL 加密的 Beeline 连接字符串

```
[alice@hadoop01 ~]$ beeline
Beeline version 0.13.1 by Apache Hive
beeline> !connect jdbc:hive2://hive.example.com:10000/default;principal=hive/hiv
e.example.com@EXAMPLE.COM;sasl.qop=auth-conf
scan complete in 4ms
Connecting to jdbc:hive2://hive.example.com:10000/default;principal=hive/hive.ex
ample.com@EXAMPLE.COM;sasl.qop=auth-conf
Enter username for jdbc:hive2://hive.example.com:10000/default;principal=hive/hi
ve.example.com@EXAMPLE.COM;sasl.qop=auth-conf:
Enter password for jdbc:hive2://hive.example.com:10000/default;principal=hive/hi
ve.example.com@EXAMPLE.COM;sasl.qop=auth-conf:
Connected to: Apache Hive (version 0.13.1)
Driver: Hive JDBC (version 0.13.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hive.example.com> show tables;
+------------+--+
|  tab_name  |
+------------+--+
| sample_07  |
| sample_08  |
+------------+--+
2 rows selected (0.261 seconds)
0: jdbc:hive2://hive.example.com>
```

如果已经配置 Hive 使用基于用户名 / 密码的身份验证，如 LDAP/Active Directory，那么 Hive 就不会再使用 SASL 进行安全连接了。这意味着需要另外一种方法启用加密。从 Hive 0.13 开始，可以配置 Hive 0.13 或更新的版本，以使用 TLS/SSL 进行加密。配置 Hive 使用 TLS 之前，需要将服务器的私钥和证书放入一个 Java keystore 文件。假定已经有了一个包含私钥和证书的 PKCS12 文件，则可以按照示例 11-32 所示的步骤将其导入 Java keystore。Hive 要求私钥的口令必须设置成与 keystore 的口令相同的值。示例中，通过在命令行设置 `-deststorepass` 和 `-destkeypass` 满足这个要求。此外，还为要导入的密钥 / 证书提供 `-srcalias` 参数。



关于 Hive 版本的注意事项

仅 Hive 0.12.0 或更新版本能够设置 SASL QOP 属性，对 TLS 加密的支持则要求 Hive 0.13.0 或更新版本。

示例 11-32　导入 PKCS12 私钥到 Java keystore

```
[root@hive ~]# mkdir /etc/hive/ssl
[root@hive ~]# keytool -v -importkeystore \
  -srckeystore /etc/pki/tls/private/hive.example.com.p12 -srcstoretype PKCS12 \
  -destkeystore /etc/hive/ssl/hive.example.com.keystore -deststoretype JKS \
  -deststorepass secret -srcalias hive.example.com -destkeypass secret
Enter source keystore password:
[Storing /etc/hive/ssl/hive.example.com.keystore]
[root@hive ~]# chown -R hive:hive /etc/hive/ssl
[root@hive ~]# chmod 400 /etc/hive/ssl/*
[root@hive ~]# chmod 700 /etc/hive/ssl
```

创建 Java keystore 之后，可以配置 Hive。如示例 11-33 所示，设置 `hive-site.xml` 文件中的配置项。

示例 11-33　配置 HiveServer2 以使用 TLS 进行加密

```
<property>
  <name>hive.server2.use.SSL</name>
  <value>true</value>
</property>
<property>
  <name>hive.server2.keystore.path</name>
  <value>/etc/hive/ssl/hive.example.com.keystore</value>
</property>
<property>
  <name>hive.server2.keystore.password</name>
  <value>secret</value>
</property>
```



使用 Kerberos 进行身份验证时，无法配置 TLS。如果在使用 Kerberos 进行身份验证，那么可以使用 SASL QOP 进行加密，否则使用 TLS。

最后，需要在客户端将 `ssl=true` 添加到 JDBC URL，以启用 TLS。如果证书不是由中央认证中心签发的，则还需要在 JDBC URL 中指定一个信任库。配置 Hive 使用 LDAPS 时，我们曾创建一个信任库，此处可以通过复制信任库文件到客户端服务器，并设置 JDBC URL 中的 `sslTrustStore` 和 `trustStorePassword` 参数复用该信任库。示例 11-34 在 Beeline 中使用 TLS 进行加密。

示例 11-34　使用 TLS 的 Beeline 连接字符串

```
[alice@hadoop01 ~]$ beeline
Beeline version 0.13.1 by Apache Hive
beeline> !connect jdbc:hive2://hive.example.com:10000/default;ssl=true;sslTrustS
tore=/etc/pki/java/hive.truststore;trustStorePassword=secret
scan complete in 3ms
Connecting to jdbc:hive2://hive.example.com:10000/default;ssl=true;sslTrustStore
=/etc/pki/java/hive.truststore;trustStorePassword=secret
Enter username for jdbc:hive2://hive.example.com:10000/default;ssl=true;sslTrust
Store=/etc/pki/java/hive.truststore;trustStorePassword=secret alice
Enter password for jdbc:hive2://hive.example.com:10000/default;ssl=true;sslTrust
Store=/etc/pki/java/hive.truststore;trustStorePassword=secret **********
Connected to: Apache Hive (version 0.13.1)
Driver: Hive JDBC (version 0.13.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hive.example.com> show tables;
+------------+--+
|  tab_name  |
+------------+--+
| sample_07  |
| sample_08  |
+------------+--+
2 rows selected (0.261 seconds)
0: jdbc:hive2://hive.example.com>
```

11.8 WebHDFS/HttpFS

Hadoop 有两种方法对 HDFS 开放 REST 接口：WebHDFS 和 HttpFS。两种系统都使用了同样的 API，因此同样的客户端能够使用这两种中的任何一种，区别在于部署方式以及数据的访问方式。WebHDFS 实际上并不是个独立的服务，它运行在 NameNode 和 DataNode 上，不适合不能直接访问集群的用户。实际上，WebHDFS 最常被用来提供版本独立的批量访问功能，例如分布式复制命令 DistCp。示例 5-10 给出了 WebHDFS 的配置示例。

相比之下，HttpFS 作为一个网关服务运行，类似 HBase REST 网关。配置 HttpFS 身份验证的第一步是配置 HttpFS，以使用 Kerberos 对 HDFS 进行身份验证。

```
<property>
  <name>httpfs.hadoop.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>httpfs.hadoop.authentication.kerberos.principal</name>
  <value>httpfs/httpfs.example.com@EXAMPLE.COM</value>
</property>
<property>
  <name>httpfs.hadoop.authentication.kerberos.keytab</name>
  <value>httpfs.keytab</value>
</property>
```

接下来，设置 HttpFS 服务器验证客户端身份的方式。依然使用 Kerberos，这将在 SPNEGO 上实现。

```
<property>
  <name>httpfs.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>httpfs.authentication.kerberos.principal</name>
  <value>HTTP/httpfs.example.com@EXAMPLE.COM</value>
</property>
<property>
  <name>httpfs.authentication.kerberos.keytab</name>
  <value>httpfs.keytab</value>
</property>
<property>
  <name>httpfs.authentication.kerberos.name.rules</name>
  <value>DEFAULT</value>
</property>
```

最后，需要配置 HttpFS 允许 Hue 用户模拟其他用户的身份，这可以通过典型的代理用户设置实现。例如，下列配置将允许 `hue` 用户模拟来自任意主机任意组的用户。

```
<property>
  <name>httpfs.proxyuser.hue.hosts</name>
  <value>*</value>
</property>
<property>
  <name>httpfs.proxyuser.hue.groups</name>
  <value>*</value>
</property>
```

11.9 小结

本章深入介绍了客户端如何访问 Hadoop 集群，从而使用其提供的众多服务和存储的数据。显而易见，由于提供给客户端的访问点数量众多，保证这种访问的安全是非常艰巨的任务。但有一个关键点始终贯穿其中：客户端必须服从已制定的认证和授权方法，例如 Kerberos 和 LDAP 提供的。

本章还讲述了用户如何使用 Sqoop、Hive、Impala、WebHDFS 和 HttpFS 从集群获取数据。虽然 Hadoop 生态系统本身已经发展多年，但商业智能的庞大生态系统 ETL 也有相应发展，其他与 Hadoop 交互的相关工具亦是如此。因此，对于管理员而言，牢牢掌握平台的数据提取功能以及保护其安全的模式是至关重要的。

第12章

Cloudera Hue

Hue 是一个 Web 应用，它为很多 Hadoop 生态系统项目提供侧重于终端用户的接口。Hadoop 配置了 Kerberos 认证后，Hue 也必须配置 Kerberos 证书，以正常访问 Hadoop。用户可以通过配置 `hue.ini` 文件中的如下参数启用 Kerberos。

`hue_principal`
: Hue 的 Kerberos 主体名称，包括 Hue 服务器的完全限定域名。

`hue_keytab`
: 包含 Hue 的服务证书的 Kerberos keytab 文件路径。

`kinit_path`
: Kerberos `kinit` 命令的路径。

`reinit_frequency`
: Hue 更新其 Kerberos 票据的频率（秒）。

这些配置应当放在 `hue.ini` 文件顶层 `[desktop]` 部分的 `[[kerberos]]` 小节。Hue kerberos 配置如示例 12-1 所示。

示例 12-1　配置 Hue 中的 Kerberos

```
[desktop]
[[kerberos]]
hue_principal=hue/hue.example.com@EXAMPLE.COM
hue_keytab=/etc/hue/conf/hue.keytab
reinit_frequency=3600
```

Hue 有自己的一套认证后端，能够对 Hadoop 和其他使用 Kerberos 的项目进行身份验证。为了代表其他用户执行动作，Hadoop 必须配置为信任 Hue 服务。这可以通过配置 Hadoop

的代理用户 / 用户模拟功能实现。需要对 Hue 可以运行的主机和 Hue 可以模拟的用户组进行设置，任何一个值都可以设为 `*`，分别表示启用来自所有主机或所有组的身份模拟。示例 12-2 展示了从主机 `hue.example.com` 以 `hadoop-users` 组的用户身份访问 Hadoop 时，如何使 Hue 进行用户模拟。

示例 12-2　在 core-site.xml 中为 Hadoop 配置 Hue 用户模拟

```
<property>
  <name>hadoop.proxyuser.hue.hosts</name>
  <value>hue.example.com</value>
</property>
<property>
  <name>hadoop.proxyuser.hue.groups</name>
  <value>hadoop-users</value>
</property>
```

HBase 和 Hive 使用 Hadoop 的模拟配置，但 Oozie 必须单独配置。如果想从 Hue 中使用 Oozie，必须在 `oozie-site.xml` 文件中设置 `oozie.service.ProxyUserService.proxyuser.<user>.hosts` 和 `oozie.service.ProxyUserService.proxyuser.<user>.groups` 属性。示例 12-3 展示了从主机 `hue.example.com` 以 `hadoop-users` 组的用户身份访问 Oozie 时，如何使 Hue 进行用户模拟。

示例 12-3　在 oozie-site.xml 中为 Oozie 配置 Hue 用户模拟

```
<property>
  <name>oozie.service.ProxyUserService.proxyuser.hue.hosts</name>
  <value>hue.example.com</value>
</property>
<property>
  <name>oozie.service.ProxyUserService.proxyuser.hue.groups</name>
  <value>hadoop-users</value>
</property>
```

如果使用 Hue 的搜索应用，还需要在 Solr 中启用用户模拟。具体方法是设置 `/etc/default/solr` 文件中的 `SOLR_SECURITY_ALLOWED_PROXYUSERS`、`SOLR_SECURITY_PROXYUSER_<user>_HOSTS` 和 `SOLR_SECURITY_PROX YUSER_<user>_GROUPS` 环境变量。示例 12-4 展示了从主机 `hue.example.com` 以 `hadoop-users` 组的用户身份访问 Solr 的用户模拟配置启用。

示例 12-4　在 /etc/default/solr 中为 Solr 配置 Hue 用户模拟

```
SOLR_SECURITY_ALLOWED_PROXYUSERS=hue
SOLR_SECURITY_PROXYUSER_hue_HOSTS=hue.example.com
SOLR_SECURITY_PROXYUSER_hue_GROUPS=hadoop-users
```

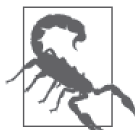
12.1　Hue HTTPS

默认情况下，Hue 运行在明文 HTTP 上，这适用于概念证明或者客户端和 Hue 之间的网络完全可信的情况。然而对于大部分环境，强烈推荐配置 Hue 使用 HTTPS。这在客户端和 Hue 之间的网络不完全受信任时尤其重要，因为大部分 Hue 的认证后端支持通过浏览器表单输入用户名和密码。

幸运的是，在 Hue 中配置 HTTPS 很容易，只需配置 `ssl_certificate` 和 `ssl_private_key`，这两项都在 `hue.ini` 文件的 `desktop` 部分。二者都必须是 PEM 格式，而且私钥不能用口令加密，如示例 12-5 所示。

示例 12-5　配置 Hue 使用 HTTPS

```
[desktop]
ssl_certificate=/etc/hue/conf/hue.crt
ssl_private_key=/etc/hue/conf/hue.pem
```



Hue 目前不支持使用受口令保护的私钥。这意味着尽最大可能保护 Hue 的私钥是很重要的。确保密钥被 hue 用户所有并只能被其所有者读取（如 `chmod 400 /etc/hue/conf/hue.pm`）。也可以在文件系统中配置文件系统级加密，按照 9.2.6 节的方法存储私钥。如果 Hue 部署在拥有被 TLS/SSL 保护的其他资源的服务器上，建议发放一个仅供 Hue 使用的证书。

12.2　Hue身份验证

Hue 拥有可插拔的身份验证框架，其中包含很多身份验证后端。默认的身份验证后端使用支持数据库中存储的用户名密码私有列表。后端的配置需要将 `[desktop]` 节 `[[auth]]` 小节下的 `backend` 属性设置为 `desktop.auth.backend.AllowFirstUserDjangoBackend`。示例 12-6 明确设置了后端的 `hue.ini` 文件。这个设置是默认的，不必费心。

示例 12-6　配置默认的 Hue 身份验证后端

```
[desktop]
[[auth]]
backend=desktop.auth.backend.AllowFirstUserDjangoBackend
```

Hue 还支持使用 Kerberos/SPNEGO、LDAP、PAM 和 SAML 进行身份验证。此处不一一介绍，要了解更多信息请参考 `config_help` 命令。[1]

12.2.1　SPNEGO后端

简单和受保护的 **GSSAPI** 协商机制（**SPNEGO**）[2] 是一个允许客户端和服务端协商选择身份验证技术的 GSSAPI 伪机制。客户端想在远程服务器上进行身份验证，但双方事先不知道对方支持的认证协议时，即可使用 SPNEGO。SPNEGO 最常用的地方是 Microsoft 最先提出的 HTTP 协商协议。[3]

Hue 只支持使用 Kerberos V5 作为底层机制的 SPNEGO，这尤其意味着，Hue 无法与 **Microsoft NT LAN Manager**（**NTLM**）协议一起使用。为 SPNEGO 配置 Hue 时，需要将

注 1：根据 Hue 的安装方式，可以执行 `/usr/share/hue/build/env/bin/hue config_help` 或 `/opt/cloudera/parcels/CDH/lib/hue/build/env/bin/hue config_help` 以执行 config_help 命令。

注 2：SPNEGO 伪机制的描述请参考 RFC 4178（http://tools.ietf.org/html/rfc4178）。

注 3：详见 Microsoft 的 MSDN 文章（http://msdn.microsoft.com/en-us/library/ms995329.aspx）。

Hue 身份验证后端设为 SpnegoDjangoBackend（示例 12-7），同时将 KRB5_KTNAME 环境变量设为含有 HTTP/< 完全限定域名 >@<REALM> 主体密钥的 keytab 文件位置。如果在服务器 hue.example.com 上手动启动 Hue，并且 keytab 文件位于 /etc/hue/conf/hue.keytab，那么应当如示例 12-8 所示启动 Hue。

示例 12-7　配置 SPNEGO Hue 身份验证后端

```
[desktop]
[[auth]]
backend=desktop.auth.backend.SpnegoDjangoBackend
```

示例 12-8　配置 KRB5_KTNAME 并人工启动 Hue

```
[hue@hue ~]$ export KRB5_KTNAME=/etc/hue/conf/hue.keytab
[hue@hue ~]$ ${HUE_HOME}/build/env/bin/supervisor
```

要使用 SPNEGO，还需要在桌面上拥有一个 TGT（例如通过运行 kinit 获得），并且需要使用支持 SPNEGO 的浏览器。Internet Explorer 和 Safari 都不需要额外配置就能支持 SPNEGO。如果使用 Firefox，首先必须将进行身份验证所用的服务器名或域名添加到受信 URI 列表。具体方法是，在 URL 栏键入 about:config，搜索 network.negotiate-auth.trusted-uris，然后更新该项使其包含该服务器名或域名。例如，如果想在 example.com 域的任意服务器中都支持 SPNEGO，那么需要设置 network.negotiate-auth.trusted-uris=example.com。如果尝试连接 Hue 时看到 401 Unauthorized 信息，那么可能没有在 Firefox 中正确配置受信 URI。

12.2.2　SAML后端

Hue 还支持使用安全断言置标语言（**Security Assertion Markup Language**，**SAML**）标准进行单点登录（**SSO**）。SAML 工作原理是，将服务提供者（**Service Provider**，**SP**）与身份提供者（**Identity Provider**，**IdP**）隔离。请求访问某个资源时，SP 会重定向到 IdP，并在那里进行身份验证；IdP 随后会向负责授予目标资源访问权限的 SP 传递一个验证身份的断言。维基百科关于 SAML 的文章（http://bit.ly/1GFpLur）里有更多细节，包括一个展示 SAML 过程步骤的图示。

配置 SAML 身份验证后端之后，Hue 就会作为服务提供者重定向到身份提供者，以进行身份验证。配置 Hue 使用 SAML 比使用其他身份验证后端更为复杂。Hue 必须与一个第三方身份提供者进行交互，因此其中的一些细节就会取决于使用的是哪个身份提供者。另外，Hue 与一些使用 SAML 所需的依赖不兼容，因此先根据示例 12-9 中的步骤安装所需依赖。

示例 12-9　安装 SAML 身份验证后端的依赖

```
[root@hue ~]# yum install swig openssl openssl-devel gcc python-devel
Loaded plugins: fastestmirror, priorities
Loading mirror speeds from cached hostfile
 * base: mirror.hmc.edu
 * extras: mirrors.unifiedlayer.com
 * updates: mirror.pac-12.org
...
Complete!
```

```
[root@hue ~]# yum install xmlsec1 xmlsec1-openssl
Loaded plugins: fastestmirror, priorities
Loading mirror speeds from cached hostfile
 * base: mirror.hmc.edu
 * extras: mirrors.unifiedlayer.com
 * updates: mirror.pac-12.org
...
Complete!
[root@hue ~]# $HUE_HOME/build/env/bin/pip install --upgrade setuptools
Downloading/unpacking setuptools from https://pypi.python.org/packages/sourc
e/s/setuptools/setuptools-7.0.tar.gz#md5=6245d6752e2ef803c365f560f7f2f940
  Downloading setuptools-7.0.tar.gz (793Kb): 793Kb downloaded
...
Successfully installed setuptools
Cleaning up...
[root@hue ~]# $HUE_HOME/build/env/bin/pip install -e \
  git+https://github.com/abec/pysaml2@HEAD#egg=pysaml2
Obtaining pysaml2 from git+https://github.com/abec/pysaml2@HEAD#egg=pysaml2
  Updating ./build/env/src/pysaml2 clone (to HEAD)
...
Successfully installed pysaml2 m2crypto importlib WebOb
Cleaning up...
[root@hue ~]# $HUE_HOME/build/env/bin/pip install -e \
  git+https://github.com/abec/djangosaml2@HEAD#egg=djangosaml2
Obtaining djangosaml2 from git+https://github.com/abec/djangosaml2@HEAD#egg=
djangosaml2
  Cloning https://github.com/abec/djangosaml2 (to HEAD) to ./build/env/src/d
jangosaml2
...
Successfully installed djangosaml2
Cleaning up...
```

如上安装一些开发工具，随后就会安装 SAML 工作所需的 Python 模块。安装完这些依赖之后，需要从身份提供者处下载元数据文件，具体的细节取决于使用的是何种身份提供者。对于 Shibboleth 身份提供者，可以使用 `curl` 将 metadata 下载到 `/etc/hue/saml/metadata.xml`。

```
[root@hue ~]# mkdir /etc/hue/saml
[root@hue ~]# curl -k -o /etc/hue/saml/metadata.xml \
   https://idp.example.com:8443/idp/shibboleth
```

还需要一个证书和私钥对请求进行签名。这个密钥必须被身份提供者所信任，能够对请求进行签名，因此可能不能仅复用启用 Hue 的 HTTPS 时所用的密钥和证书。就本文而言，我们假定已经创建了密钥和证书，并且分别存放在 `/etc/hue/saml/key.pem` 和 `/etc/hue/saml/idp.pem` 文件中，剩下的就是配置 Hue 本身了。示例 12-10 给出了 `/etc/hue/conf/hue.ini` 文件中的相关部分。

示例 12-10　配置 SAML Hue 身份验证后端

```
[desktop]
[[auth]]
backend=libsaml.backend.SAML2Backend
```

```
[libsaml]
xmlsec_binary=/usr/bin/xmlsec1
create_users_on_login=true
metadata_file=/etc/hue/saml/metadata.xml
key_file=/etc/hue/saml/key.pem
cert_file=/etc/hue/saml/idp.pem
```

有一些额外可选的配置参数可以在 `saml` 配置组里设置，完整的配置参数列表如下。

`xmlsec_binary`
: xmlsec1 程序的路径，该程序用于签名、验证、加密和解密 SAML 请求和断言。典型值为 `/usr/bin/xmlsec1`。

`create_users_on_login`
: 登录时创建 Hue 用户，可以是 `true` 或者 `false`。

`required_attributes`
: Hue 必需的、从 IdP 获取的属性。其值是逗号分隔的属性列表。例如：`uid, email`。

`optional_attributes`
: Hue 可以处理的从 IdP 获取的属性。其值是逗号分隔的属性列表，处理方式与 `required_attributes` 一样。

`metadata_file`
: 从 IdP 获取的元数据 XML 文件的路径。该文件必须能够被 `hue` 用户读取。

`key_file`
: PEM 格式的密钥文件。

`cert_file`
: PEM 格式的 X.509 证书。

`user_attribute_mapping`
: 将从 IdP 接收到的属性（在 `required_attributes`、`optional_attributes` 和 IdP 配置中指定的）映射到 Hue 用户属性。例如：`{`*`uid:`*`'username', email: `*`email`*`}`。

`authn_requests_signed`
: 对身份验证请求进行签名。其值可以是 `true` 或 `false`，检查 IdP 文档，确认是否需要进行该项设置。

`logout_requests_signed`
: 对登出请求进行签名。其值可以是 `true` 或 `false`。检查 IdP 文档，确认是否需要进行该项设置。

12.2.3　LDAP后端

最后一种身份验证方式是使用 LDAP/Active Directory 验证用户名和口令。有两种配置 Hue 的 LDAP 后端的方法，第一种方法是进行 LDAP 搜索，找到用户的标识名（DN），然后使用 DN 绑定到 LDAP；第二种方法是向 Hue 提供一个填写了用户名的 DN 模式，之后不用

搜索即可绑定。

配置 Hue 使用搜索绑定时，必须将 search_bind_authentication 设为 true，并设置 desktop 节 ldap 小节中的 ldap_url 和 base_dn，还必须设置 desktop 节 ldap 小节下 users 小节中的 user_filter 和 user_name_attr，还应当设置 desktop 节 ldap 小节下 groups 小节中的 group_filter、group_name_attr 和 group_member_attr，这样才能将 LDAP 组导入为 Hue 组。

示例 12-11 展示了使用搜索绑定进行 LDAP 身份验证的 hue.ini 配置文件片段。示例中，LDAP 服务器运行在 ldap.example.com 的 LDAPS 上。该 LDAP 服务器将用户和组存储在一个基准 DN cn=accounts, dc=example,dc=com 之下。最后，用户账户在 objectClass=posixaccount 中，用户组在 objectClass=posixgroup 中。对于所有 LDAP 相关配置的完整描述见表 12-1。

示例 12-11　配置使用搜索绑定的 LDAP Hue 身份验证后端

```
[desktop]
[[auth]]
backend=desktop.auth.backend.LdapBackend

[[ldap]]
ldap_url=ldaps://ldap.example.com
base_dn="cn=accounts,dc=example,dc=com"
search_bind_authentication=true
ldap_cert=/etc/hue/conf/ca.crt
use_start_tls=false
create_users_on_login=true
[[[users]]]
user_filter="objectClass=posixaccount"
user_name_attr="uid"

[[[groups]]]
group_filter="objectClass=posixgroup"
group_name_attr="cn"
group_member_attr="member"
```

如果倾向于使用直接绑定，那么必须将 search_bind_authentication 设为 false，并且根据使用的是 Active Directory 还是其他 LDAP 提供者分别设置 nt_domain 或者 ldap_username_pattern。仍然必须配置搜索相关的配置项（例如 user_filter、user_name_attr 等），因为从 LDAP 同步用户和组的时候会用到。如果想要使用与之前相同的服务器配置，但是用直接绑定而不是搜索绑定，那么与示例 12-12 类似。再次强调，LDAP 配置参数的完整列表见表 12-1。

示例 12-12　配置使用直接绑定的 LDAP Hue 身份验证后端

```
[desktop]
[[auth]]
backend=desktop.auth.backend.LdapBackend

[[ldap]]
ldap_url=ldaps://ldap.example.com
```

```
base_dn="cn=accounts,dc=example,dc=com"
search_bind_authentication=false
ldap_username_pattern="uid=<username>,cn=users,cn=accounts,dc=example,dc=com"
ldap_cert=/etc/hue/conf/ca.crt
use_start_tls=false
create_users_on_login=true

[[[users]]]
user_filter="objectClass=posixaccount"
user_name_attr="uid"

[[[groups]]]
group_filter="objectClass=posixgroup"
group_name_attr="cn"
group_member_attr="member"
```

表12-1：Hue LDAP身份验证的配置属性

节	属性	描述
desktop.auth	backend	使用的身份验证后端（设为 desktop.auth.backend.LdapBackend）
desktop.ldap	ldap_url	LDAP 服务器 URL（对于安全 LDAP 使用 ldaps://）
desktop.ldap	base_dn	用于 LDAP 搜索的基本 LDAP 标识名
desktop.ldap	bind_dn	搜索 LDAP 时绑定的标识名；仅当 LDAP 服务器上的匿名搜索被禁用时需要
desktop.ldap	bind_password	bind_dn 用户的密码；仅当 LDAP 服务器上的匿名搜索被禁用时需要
desktop.ldap	create_users_on_login	设为 true，在首次登录时创建用户；如果设为 false，管理员必须手动向 Hue 添加用户，之后这些用户才能使用其 LDAP 凭证登录
desktop.ldap	search_bind_authentication	设为 true，使用搜索绑定；设为 false，启用直接绑定
desktop.ldap	ldap_username_pattern	从用户名创建标识名的模式——必须包含字符串 <*username*>，该值会被用户的用户名替代，从而创建最终的 DN；仅当配置 Hue 为直接绑定时（即 search_bind_authentication=false）使用
desktop.ldap	nt_domain	Active Directory 服务器的 NT 域；仅当配置 Hue 为直接绑定时（即 search_bind_authentication=*false*）使用
desktop.ldap	ldap_cert	验证 LDAP 服务器证书的 CA 证书位置
desktop.ldap	use_start_tls	设为 true，使用 StartTLS；ldap_url 为 ldaps://URL 时，设为 false
desktop.ldap.users	user_filter	搜索用户时使用的基础过滤器
desktop.ldap.users	user_name_attr	LDAP 模式中的用户名属性（通常是 Active Diretory 的 sAMAccountName 和 LDAP 目录的 uid）
desktop.ldap.groups	group_filter	搜索组时使用的基础过滤器
desktop.ldap.groups	group_name_attr	LDAP 模式中的组属性（通常是 cn）
desktop.ldap.groups	group_member_attr	指定组成员的 LDAP 属性（通常是 member）

由示例 12-11 和示例 12-12 可知，将 `ldap_cert` 设置为指向一个 CA 证书，这是必需的，因为使用了 `ldaps://` 的方式配置 LDAP URL。强烈建议使用 LDAPS 或 StartTLS。使用 StartTLS 时，要用 `ldap://` 的方式配置 LDAP URL，并且将 `use_start_tls` 设为 `true`。

12.3 Hue授权

Hue 有两种用户账户：普通用户和超级用户。普通用户由基于访问控制表（ACL）的授权系统进行管理，该系统控制着哪些应用权限可用于哪些组。Hue 超级用户可以：

- 添加和删除用户
- 添加和删除组
- 给组分配权限
- 将某个用户变为超级用户
- 从 LDAP 服务器导入用户和组
- 安装查询、表和数据的样例
- 查看、提交和修改任意 Oozie 工作流、协调程序或 bundle
- 查看和修改 Sentry 权限时模拟任意用户
- 查看和修改 HDFS ACLs 时模拟任意用户



Hue 超级用户与 HDFS 超级用户不同。HDFS 超级用户是运行 NameNode 守护进程的用户，通常是 `hdfs`，并拥有列举、读取和写入任意 HDFS 文件和目录的权限。如果想从 Hue 上执行 HDFS 超级用户操作，需要添加一个与 HDFS 超级用户同名的用户。或者可以设置 HDFS 超级组，以给一组用户分配 HDFS 超级用户权限。参考 6.1 节。

每个 Hue 应用定义了用户可以执行的一个或多个操作，授权控制通过为每个操作配置 ACL 实现，ACL 列出了可以执行该操作的组。每个应用都有一个名为“启动该应用”的操作，这决定了哪些用户可以运行该应用。一些应用还定义了额外的可控操作。



Hue 中授予的权限仅仅是从特定 Hue 应用调用特定操作的权限。执行某个操作的用户仍然需要由被他们访问的服务进行授权。例如，一个用户可能拥有 Metastore 应用中“允许 DDL 操作”的权限，但如果没有 Sentry 数据库中的 ALL 权限，依然无法创建表。

HBase 应用定义了“允许在 HBase 应用中写入”的操作，这授予了从 HBase 应用中添加行、添加单元、编辑单元、删除行和删除单元的权限。Metastore 应用定义了“允许 DDL 操作”的操作，这授予了从 metastore 浏览器创建、编辑和删除表的权限。Oozie 应用定义了“对所有作业的 Oozie 仪表盘只读用户”，这授予了对——无论是否被共享——所有工作流、协调器和 bundle 进行只读访问的权限。Security 应用定义了“列举文件或表等对象时，让一个用户模拟另外一个用户的身份”的操作，该操作允许用户模拟其他用户，并查看该用户能访问哪些表、文件和目录。



为“列举文件或表等对象时，让一个用户模拟另外一个用户的身份”授予权限，会暴露原本不可用的信息。特别地，它允许一个没有获得授权的已登录用户模拟另一个能够查看目录中文件的用户，授予执行该操作的权限时应当谨慎。另外值得提醒的是，Hue 超级用户还能够在 Security 应用中模拟用户，因此将一个用户变为 Hue 超级用户时也应当注意。

Useradmin 应用定义了“在 User Admin 上访问个人资料页面”的操作，但该操作已被弃用，可放心忽略。

12.4 Hue SSL客户端配置

之前介绍了很多安全相关的配置，但还没有详尽介绍如何在一般情况下建立和配置 Hue，这超出了本书范围。但一个重要问题是，各种底层组件被配置成 SSL 在线加密时，如何恰当配置 Hue。如果 Hadoop、Hive 和 Impala 启用了 SSL，示例 12-13 展示了必要的配置片段。

示例 12-13　Hue SSL 配置

```
# Non-SSL configurations omitted for brevity
[beeswax]
 [[ssl]]
enabled=true
cacerts=/etc/hue/ca.cer
key=/etc/hue/host.key
cert=/etc/hue/host.cer
validate=true
[impala]
[[ssl]]
enabled=true
cacerts=/etc/hue/ca.cer
key=/etc/hue/host.key
cert=/etc/hue/host.cer
validate=true
```

Hive 和 Impala 配置中，`validate` 选项都会指定是否要求 Hue 验证哪些服务提供的证书是由配置的证书链中的机构签发的。

除示例之外，环境变量 `REQUESTS_CA_BUNDLE` 需要指向 SSL 证书链（PEM 格式）文件所在的磁盘位置。这被 Hadoop SSL 客户端用于访问 HDFS、MapReduce、YARN 和 HttpFS。

12.5 小结

本章详细讨论了允许终端用户通过集中式网络控制台访问各种生态系统组件时，Hue 的重要地位。在 Hue 的帮助下，用户只需登录网络控制台一次，剩下的集群操作都通过 Hue 系统用户的身份模拟执行。

本章对数据安全组件的讨论到此结束，接下来通过一些实际场景的案例学习，整合之前学到的知识。

第四部分

综合应用



第13章

案例分析

本章将介绍两个覆盖了书中很多安全主题的案例分析。首先，我们看一下如何在多重任务处理环境中使用 Sentry，以控制 SQL 访问数据。这在深入讨论一个更详细的案例前是个很好的热身，接下来的案例展示了一个自定义 HBase 应用以及各种安全特性。

13.1 案例分析：Hadoop数据仓库

大数据和 Hadoop 的主要好处之一就是，可以汇集很多不同的数据集解决独特问题。随之而来的是跨越多个业务线的不同类型的用户。该案例分析中，我们将看看在包含多条业务线、多个数据所有者以及不同分析员的环境下，如何用 Sentry 提供强大的 Hive 和 Impala 数据授权。

首先列出这个案例分析的假设。

- 环境包含 3 条业务线，我们将其称为 `lob1`、`lob2` 和 `lob3`。
- 每条业务线拥有分析员和管理员：
  — 分析员由组 `lob1grp`、`lob2grp` 和 `lob3grp` 定义；
  — 管理员由组 `lob1adm`、`lob2adm` 和 `lob3adm` 定义；
  — 管理员也在分析员组中。
- 每条业务线均需要在HDFS中拥有自己的沙箱区进行即席分析,并上传自服务的数据资源。
- 每条业务线拥有自己的管理员，控制对各自沙箱的访问权限。
- Hive 仓库中的数据是 IT 管理的，意味着只有非交互的 ETL 用户会添加数据。
- 只有 Hive 管理员会在 Hive 仓库中创建新的对象。
- Hive 仓库使用默认的 HDFS 路径 `/user/hive/warehouse`。
- 已经为集群设置 Kerberos。
- 环境中已经建立 Sentry。

- HDFS 已经启用扩展 ACL。
- HDFS 的默认权限掩码设置为 007。

13.1.1 环境搭建

有了基本假设后，需要在 HDFS 中为 Sentry 准备必要的目录。首先要做的就是锁定 Hive 仓库目录。启用 Sentry 时，HiveServer2 的身份模拟会被禁用，所以只有 `hive` 组具有访问权限（包括 `hive` 和 `impala` 用户）。我们需要做的如下所示：

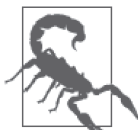
```
[root@server1 ~]# kinit hive
Password for hive@EXAMPLE.COM:
[root@server1 ~]# hdfs dfs -chmod -R 0771 /user/hive/warehouse
[root@server1 ~]# hdfs dfs -chown -R hive:hive /user/hive/warehouse
[root@server1 ~]#
```

正如前面假设所言，每条业务线都需要一个沙箱区域。需要创建一个 `/data/sandbox` 作为所有沙箱的根目录，然后在其中创建相应的结构：

```
[root@server1 ~]# kinit hdfs
Password for hdfs@EC2.INTERNAL:
[root@server1 ~]# hdfs dfs -mkdir /data
[root@server1 ~]# hdfs dfs -mkdir /data/sandbox
[root@server1 ~]# hdfs dfs -mkdir /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -mkdir /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -mkdir /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -chmod 770 /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -chmod 770 /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -chmod 770 /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -chgrp lob1grp /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -chgrp lob2grp /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -chgrp lob3grp /data/sandbox/lob3
[root@server1 ~]#
```

基本的目录结构建好之后，需要开始思考，要用什么支持 `Hive` 和 `Impala` 访问沙箱，毕竟这些沙箱是所有用户要进行即席分析的地方。`hive` 和 `impala` 用户都需要访问这些目录，所以下面要创建 HDFS 扩展的 ACL（HDFS-extended ACL），以允许 `hive` 组的完全访问。

```
[root@server1 ~]# hdfs dfs -setfacl -m default:group:hive:rwx /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m default:group:hive:rwx /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m default:group:hive:rwx /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -setfacl -m group:hive:rwx /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m group:hive:rwx /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m group:hive:rwx /data/sandbox/lob3
[root@server1 ~]#
```



记住，默认 ACL 只适用于目录，它仅规定了复制到新的子目录和文件中的 ACL。由于这个原因，父目录依然需要一个常规访问 ACL。

接下来需要确保，无论谁创建新文件，都能保持预期的访问权限不变。如果放任现在的权

限不管，那么 `hive` 或 `impala` 用户新创建的目录和文件就可能被业务线中的分析员和管理员访问。为了解决这个问题，向扩展 ACL 添加以下组：

```
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob1grp:rwx \
  /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob1adm:rwx \
  /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob2grp:rwx \
  /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob2adm:rwx \
  /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob3grp:rwx \
  /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -setfacl -m default:group:lob3adm:rwx \
  /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -setfacl -m group:lob1grp:rwx /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m group:lob1adm:rwx /data/sandbox/lob1
[root@server1 ~]# hdfs dfs -setfacl -m group:lob2grp:rwx /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m group:lob2adm:rwx /data/sandbox/lob2
[root@server1 ~]# hdfs dfs -setfacl -m group:lob3grp:rwx /data/sandbox/lob3
[root@server1 ~]# hdfs dfs -setfacl -m group:lob3adm:rwx /data/sandbox/lob3
[root@server1 ~]#
```

所有扩展 ACL 已设置，查看其中之一：

```
[root@server1 ~]# hdfs dfs -getfacl -R /data/sandbox/lob1
# file: /data/sandbox/lob1
# owner: hdfs
# group: lob1grp
user::rwx
group::rwx
group:hive:rwx
group:lob1adm:rwx
group:lob1grp:rwx
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:hive:rwx
default:group:lob1adm:rwx
default:group:lob1grp:rwx
default:mask::rwx
default:other::---
[root@server1 ~]#
```

我们已经处理了所有租用集群的用户，还要确认也在 HDFS 中为非交互的 ETL 用户创建了可用空间：

```
[root@server1 ~]# hdfs dfs -mkdir /data/etl
[root@server1 ~]# hdfs dfs -chown etluser:hive /data/etl
[root@server1 ~]# hdfs dfs -chmod 770 /data/etl
[root@server1 ~]# hdfs dfs -setfacl -m default:group:hive:rwx /data/etl
[root@server1 ~]# hdfs dfs -setfacl -m group:hive:rwx /data/etl
[root@server1 ~]# hdfs dfs -setfacl -m default:user:etluser:rwx /data/etl
[root@server1 ~]# hdfs dfs -setfacl -m user:etluser:rwx /data/etl
```

```
[root@server1 ~]# hdfs dfs -getfacl /data/etl
# file: /data/etl
# owner: etluser
# group: hive
user::rwx
user:etluser:rwx
group::rwx
group:hive:rwx
mask::rwx
other::---
default:user::rwx
default:user:etluser:rwx
default:group::rwx
default:group:hive:rwx
default:mask::rwx
default:other::---
[root@server1 ~]#
```

下一步是开始在 Hive 中使用 `beeline` 命令行执行一些管理任务。我们将使用 `hive` 用户，因为默认情况下它是一个 Sentry 管理员，因此可以创建策略。



你可以使用属性文件为 `beeline` 指定连接信息。这比记住语法或者查找 bash 历史记录容易得多。

我们要使用的 `beeline.properties` 文件如示例 13-1 所示。注意，用户名和口令是实际认证所必需的，但空着，因为启用了 Kerberos。

示例 13-1　beeline.properties 文件

```
ConnectionURL=jdbc:hive2://server1.example.com:10000/;principal=
  hive/server1.example.com@EXAMPLE.COM
ConnectionDriverName=org.apache.hive.jdbc.HiveDriver
ConnectionUserName=.
ConnectionPassword=.
```

```
[root@server1 ~]# kinit hive
Password for hive@EXAMPLE.COM:
[root@server1 ~]# beeline
...
beeline> !properties beeline.properties
...
> CREATE ROLE sqladmin;
> GRANT ROLE sqladmin TO GROUP hive;
> GRANT ALL ON SERVER server1 TO ROLE sqladmin;
> CREATE DATABASE lob1 LOCATION '/data/sandbox/lob1';
> CREATE DATABASE lob2 LOCATION '/data/sandbox/lob2';
> CREATE DATABASE lob3 LOCATION '/data/sandbox/lob3';
> CREATE DATABASE etl LOCATION '/data/etl';
```

创建管理员角色和数据库之后，可以设置 Sentry 策略为 Hive 和 Implala 提供对终端用户的授权。

```
> CREATE ROLE lob1analyst;
> GRANT ROLE lob1analyst TO GROUP lob1grp;
> GRANT ALL ON DATABASE lob1 TO ROLE lob1analyst;
> CREATE ROLE lob1administrator;
> GRANT ROLE lob1administrator TO GROUP lob1adm WITH GRANT OPTION;
> GRANT ALL ON DATABASE lob1 TO role lob1administrator;
> CREATE ROLE lob2analyst;
> GRANT ROLE lob2analyst TO GROUP lob2grp;
> GRANT ALL ON DATABASE lob2 TO ROLE lob2analyst;
> CREATE ROLE lob2administrator;
> GRANT ROLE lob2administrator TO GROUP lob2adm WITH GRANT OPTION;
> GRANT ALL ON DATABASE lob2 TO ROLE lob2administrator;
> CREATE ROLE lob3analyst;
> GRANT ROLE lob3analyst TO GROUP lob3grp;
> GRANT ALL ON DATABASE lob3 TO role lob3analyst;
> CREATE ROLE lob3administrator;
> GRANT ROLE lob3administrator TO GROUP lob3adm WITH GRANT OPTION;
> GRANT ALL ON DATABASE lob3 TO ROLE lob3administrator;
> CREATE ROLE etl;
> GRANT ROLE etl TO GROUP etluser;
> GRANT ALL ON DATABASE etl TO ROLE etl;
```

列在假设里的另一个重要需求是，用户应能上传自服务文件到他们自己的沙箱。要允许用户利用 Hive 和 Impala 中的这些文件，他们还需要一些 URI 权限。我们还将继续提供写权限，使用户能够从 Hive 提取数据并导入沙箱区域，以供额外的非 SQL 分析使用。

```
> GRANT ALL ON URI 'hdfs://nameservice1/data/etl' TO ROLE etl;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob1' TO ROLE lob1analyst;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob1'
  TO ROLE lob1administrator;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob2' TO ROLE lob2analyst;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob2'
  TO ROLE lob2administrator;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob3' TO ROLE lob3analyst;
> GRANT ALL ON URI 'hdfs://nameservice1/data/sandbox/lob3'
  TO ROLE lob3administrator;
```



以上展示的 URI 路径使用了 HDFS HA 命名服务名称。如果没有配置 HA，则需要指定 NameNode 的完全限定域名，包括端口（8020）。

13.1.2　用户体验

当环境全部上线就绪，并且配备一整套 HDFS 权限和 Sentry 策略后，下面看看在这些落实的措施下，终端用户看到的内容。首先看看 `sqladmin` 角色的用户会看到什么：

```
[root@server1 ~]$ kinit hive
Password for hive@EXAMPLE.COM:
[root@server1 ~]$ beeline
...
> !properties beeline.properties
```

```
...
> SHOW DATABASES;
+----------------+
| database_name  |
+----------------+
| default        |
| etl            |
| lob1           |
| lob2           |
| lob3           |
+----------------+
> quit;
[root@server1 ~]$
```

如上所示，sqladmin 角色被允许查看我们创建的每一个数据库。这和预期的一致，因为 sqladmin 角色被授予了对 SERVER 对象的完全访问权限。接下来看看被分配到 etl 角色的用户会看到什么：

```
[root@server1 ~]$ kinit etluser
Password for etluser@EXAMPLE.COM:
[root@server1 ~]$ beeline
...
> !properties beeline.properties
...
> SHOW DATABASES;
+----------------+
| database_name  |
+----------------+
| default        |
| etl            |
+----------------+
> USE lob1;
Error: Error while compiling statement: FAILED: SemanticException
 No valid privileges (state=42000,code=40000)
> quit;
[root@server1 ~]$
```

用户这次没有看到 metastore 中的所有数据库，而只看到了包含他们具有访问权限的对象的数据库。上述例子说明，不仅用户没有访问权限的对象是对用户隐藏的，而且即便用户通过名称请求该对象，也会被拒绝。这正是我们期望的。

现在，假定 etl 数据库中的表 sample_07 需要变成对 lob1analyst 角色可用，然而要注意：并非所有列都可以共享。为此，需要创建一个只包含我们想对该角色可见的列的视图。创建该视图后，授予 lob1analyst 角色对其的访问权限：

```
[root@server1 ~]$ kinit hive
Password for hive@EXAMPLE.COM:
[root@server1 ~]$ beeline
...
> !properties beeline.properties
...
> USE etl;
> CREATE VIEW sample_07_view AS SELECT code, description, total_emp
```

```
FROM sample_07;
> GRANT SELECT ON TABLE sample_07_view TO ROLE lob1analyst;
> quit;
[root@server1 ~]$
```

完成这些工作后，可以用一个被分配到 lob1analyst 角色的用户测试访问权限：

```
[root@server1 ~]$ kinit lob1user
Password for lob1user@EXAMPLE.COM:
[root@server1 ~]$ beeline
...
> !properties beeline.properties
...
> SHOW DATABASES;
+----------------+
| database_name  |
+----------------+
| default        |
| etl            |
| lob1           |
+----------------+
> USE etl;
> SHOW TABLES;
+----------------+
|    tab_name    |
+----------------+
| sample_07_view |
+----------------+
> SELECT * FROM sample_07 LIMIT 1;
Error: Error while compiling statement: FAILED: SemanticException
 No valid privileges (state=42000,code=40000)
> quit;
[root@server1 ~]$ hdfs dfs -ls /data/etl
ls: Permission denied: user=lob1user, access=READ_EXECUTE, inode="/data/etl":
 etluser:hive:drwxrwx---:group::---,group:hive:rwx,
 default:user::rwx,default:group::---,default:group:hive:rwx,
 default:mask::rwx,default:other::---
[root@server1 ~]$
```

如上所示，lob1user 能够在列表中看到 etl 数据库。但注意，该数据库中只有 sample_07_view 对象是可见的。如我们所预期的，用户无法通过 SQL 访问或者直接的 HDFS 访问读取源表。上述例子中有一些 access denied 信息，所以查看日志文件中出现了什么。从 HiveServer2 日志开始：

```
2015-01-13 19:31:40,173 ERROR org.apache.hadoop.hive.ql.Driver: FAILED:
 SemanticException No valid privileges
org.apache.hadoop.hive.ql.parse.SemanticException: No valid privileges
        at org.apache.sentry.binding.hive.HiveAuthzBindingHook.
         postAnalyze(HiveAuthzBindingHook.java:320)
        at org.apache.hadoop.hive.ql.Driver.compile(Driver.java:457)
        at org.apache.hadoop.hive.ql.Driver.compile(Driver.java:352)
        at org.apache.hadoop.hive.ql.Driver.compileInternal
         (Driver.java:995)
        at org.apache.hadoop.hive.ql.Driver.compileAndRespond
```

```
        (Driver.java:988)
        at org.apache.hive.service.cli.operation.SQLOperation.prepare
        (SQLOperation.java:98)
        at org.apache.hive.service.cli.operation.SQLOperation.run
        (SQLOperation.java:163)
        at org.apache.hive.service.cli.session.HiveSessionImpl.
        runOperationWithLogCapture(HiveSessionImpl.java:524)
        at org.apache.hive.service.cli.session.HiveSessionImpl.
        executeStatementInternal(HiveSessionImpl.java:222)
        at org.apache.hive.service.cli.session.HiveSessionImpl.
        executeStatement(HiveSessionImpl.java:204)
        at org.apache.hive.service.cli.CLIService.executeStatement
        (CLIService.java:168)
        at org.apache.hive.service.cli.thrift.ThriftCLIService.
        ExecuteStatement(ThriftCLIService.java:316)
        at org.apache.hive.service.cli.thrift.TCLIService$Processor
        $ExecuteStatement.getResult(TCLIService.java:1373)
        at org.apache.hive.service.cli.thrift.TCLIService$Processor
        $ExecuteStatement.getResult(TCLIService.java:1358)
        at org.apache.thrift.ProcessFunction.process
        (ProcessFunction.java:39)
        at org.apache.thrift.TBaseProcessor.process(TBaseProcessor.java:39)
        at org.apache.hadoop.hive.thrift.HadoopThriftAuthBridge20S$Server
        $TUGIAssumingProcessor.process(HadoopThriftAuthBridge20S.java:608)
        at org.apache.thrift.server.TThreadPoolServer$WorkerProcess.run
        (TThreadPoolServer.java:244)
        at java.util.concurrent.ThreadPoolExecutor.runWorker
        (ThreadPoolExecutor.java:1145)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run
        (ThreadPoolExecutor.java:615)
        at java.lang.Thread.run(Thread.java:745)
Caused by: org.apache.hadoop.hive.ql.metadata.AuthorizationException:
        User lob1user does not have privileges for QUERY
        at org.apache.sentry.binding.hive.authz.HiveAuthzBinding.authorize
        (HiveAuthzBinding.java:317)
        at org.apache.sentry.binding.hive.HiveAuthzBindingHook.
        authorizeWithHiveBindings(HiveAuthzBindingHook.java:502)
        at org.apache.sentry.binding.hive.HiveAuthzBindingHook.
        postAnalyze(HiveAuthzBindingHook.java:312)
        ... 20 more
```

接下来，可以看到出现在 NameNode 审计日志中的拒绝访问的审计事件：

```
2015-01-13 20:01:15,005 INFO FSNamesystem.audit: allowed=false
  ugi=lob1user@EXAMPLE.COM (auth:KERBEROS)
ip=/10.6.9.73
 cmd=listStatus src=/data/etl dst=null perm=null
```

13.1.3 小结

这个基础的案例分析展示了，如何看待使用 Sentry 策略外加 HDFS 扩展 ACL 进行数据保护。该示例有意做得很简单，但它也说明，在多重任务处理环境下，将数据组织作为一个关键因素考虑何等重要。只有对数据如何存放在 HDFS 中具有一个清晰的概念，安全管理才会更容易。

13.2 案例分析：交互式HBase Web应用

Hadoop 的一个常见用例是搭建大规模 Web 应用。HBase 的一些特性使它成为交互式大规模应用的理想选择：

- 支持复杂对象、具有快速演化模式的灵活的数据模型
- 添加或移除集群节点时的数据自动重分区
- 集成 Hadoop 生态系统其余部分以支持事务数据的离线分析
- 行内 ACID 交易
- 对各种应用的高级授权功能

对于本书，我们最关心列表中的最后一个功能。对于交互式应用，你经常需要对哪个用户能访问哪个数据集进行控制。例如，一个类似 Twitter 的应用就具有完全公开的消息、仅限授权用户白名单的消息，以及完全私有的消息。要想在这种动态安全需求面前灵活管理授权，需要同样动态的数据库使用方式。

这个案例分析中，我们将看到一个存储和浏览网页快照的应用。这个案例分析建立在 The Kite SDK（http://kitesdk.org）提供的一个基于 HBase 的开源 Web 应用示例（https://github.com/kite-sdk/kite-spring-hbase-example），原本的示例作为部署在 OpenShift 中的一个应用，以及部署在 HBase 集群中的一个生产应用，工作在单机开发模式下。由于 `MiniHBaseCluster` 类（用于开发模式和 OpenShift 部置）的局限，我们的版本仅为生产应用，保护 HBase 集群。我们这个版本的该示例完整源代码在与本书配套的 GitHub 源码库（https://github.com/hadoop-security/kite-spring-hbase-example）中。

13.2.1 设计与架构

首先看看这个网页快照示例的架构，如图 13-1 所示。该 Web 应用部署在一个边界节点上，用户通过浏览器连接到该应用，并使用 URL 创建新快照或者查看已有快照。创建新快照时，该 Web 应用会下载网页和元数据，并将它们存储在 HBase 中。查看快照时，该 Web 应用会从 HBase 获取页面元数据和快照，并展示到浏览器。

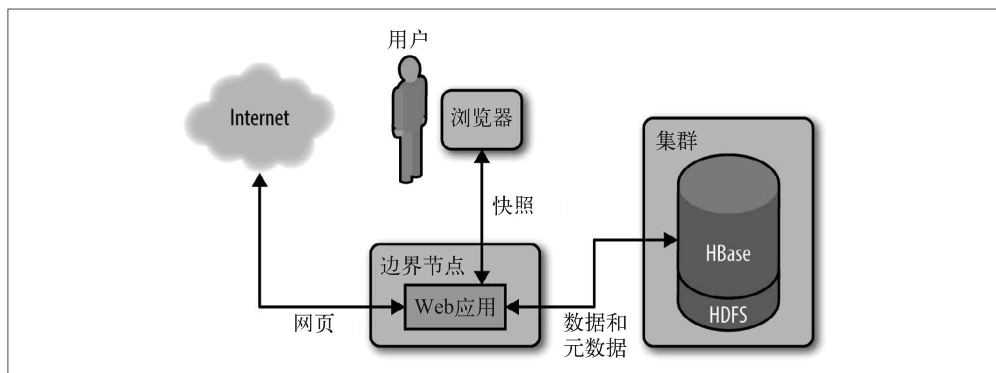


图 13-1：Web 应用架构

开始讨论安全需求之前，先看看该例中使用的数据模型。每个网页都使用一个 URL 作为唯一标识，每个快照由获取网页的时间进一步确定。数据模型中的完整字段列表见表 13-1。

表13-1：网页快照数据模型

字段	类型	描述
url	String	网页的 URL
fetchedAt	long	网页被获取的 UTC 时间
fetchTimeMs	int	获取网页所用的时间，以 ms 为单位
size	int	网页大小
title	String	HTML 页面标题（如果有）
description	String	HTML 元标签（meta tag）中的描述
keywords	List<String>	HTML 元标签中的关键字
outlinks	List<String>	该网页链接到的页面 URL
content	String	网页内容

HBase 将数据存储为多维排序映射，这意味着需要将我们记录里的字段映射到 Habse 用于排列数据的行键、列族和列限定键。对于用例而言，我们希望 HBase 的每行以 URL 和获取快照的时间为键值进行检索。为了使最近的快照排在前面，我们在行键中使用 `fetchedAt` 时间戳之前，将先使用 `Long.MAX_VALUE` 减去其值，从而实现反向排序，见图 13-2 中的 `<rev fetchedAt>`。每个字段都对应 HBase 中的一列，因此定义了从每个字段名称到列族和列限定符的映射。图 13-2 展示了行键是如何被映射的，并给出了映射到 HBase 列的字段示例。

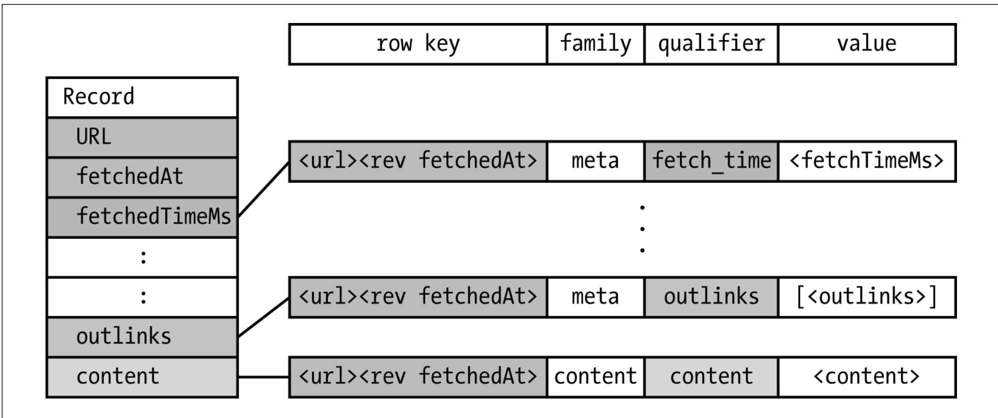


图 13-2：原始的 HBase 数据模型映射

13.2.2 安全需求

现在可以向该示例添加安全特性了。默认情况下，快照中的所有字段都能被任意用户访问。我们的用例中，想在默认情况下锁定网页内容，只在我们要求公开某个快照时才能被

访问。可以使用单元级安全并保持数据模型与之前使用的一样，但这对于用例而言可能是“杀鸡用牛刀”。相反，我们可以稍微修改一下数据模型。

特别地，我们将向模型添加一个 `contentKey` 字段，`contentKey` 将被用作存储内容的列限定符。使用用户名作为私有快照的 `contentKey`，对于公开快照则使用特殊值 `public`。现在希望在可能的不同列限定符中存储每个快照的内容。因此，将 `content` 字段的类型改为 `Map<String, String>`。图 13-3 展示了更新后的映射配置。

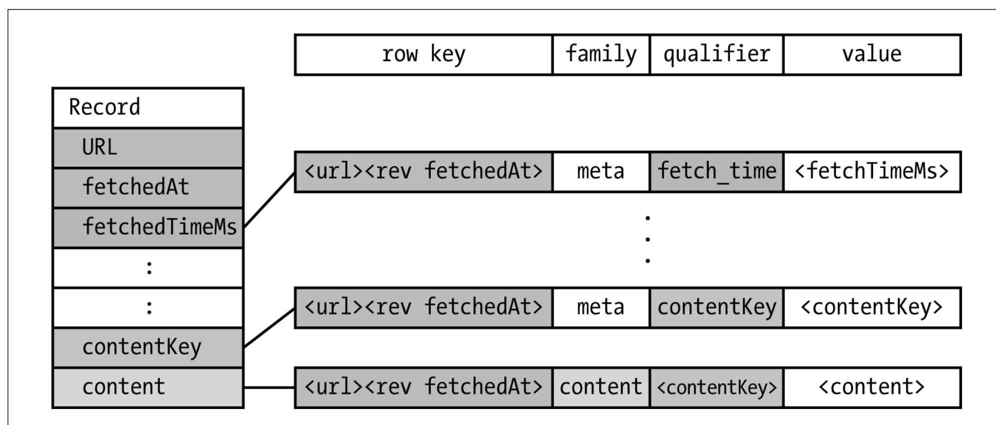


图 13-3：更新后的 HBase 数据模型映射

继续之前，先总结一下想在应用中实施的安全需求列表：

(1) 私有快照的内容只能被创建该快照的用户访问
(2) 公开快照的内容对所有用户可见
(3) 所有快照的元数据对所有用户可见
(4) 用户使用 HTTP 基本认证进行应用的身份验证
(5) 应用模拟已认证用户的身份与 HBase 通信
(6) 在 HBase 级进行强制授权

13.2.3　集群配置

有了这些需求，下面可以开始配置集群。需求 (5) 说明该应用需要与 HBase 进行身份验证。为了启用 HBase 身份验证，必须先启用 Hadoop 身份验证。要满足需求 (6)，还需要启用 HBase 授权，HBase 授权也是需求 (1) 和 (2) 的必要条件。需求 (3) 说明，要允许所有用户访问元数据字段。需求 (4) 适用于 Web 应用自身以及应用服务器，对于我们而言，使用的是 Tomcat。现在准备开始计划我们的配置步骤：

(1) 配置 Hadoop 身份验证（见 5.2.5 节）；
(2) 配置 HBase 身份验证［参考 *The Apache HBase Reference Guide*（http://hbase.apache.org/book.html）中的 Securing Apache HBase（http://hbase.apache.org/book.html#security）］；
(3) 向 `hbase-site.xml` 添加如下内容，以配置 HBase 授权：

```
<property>
  <name>hbase.coprocessor.region.classes</name>
  <value>
    org.apache.hadoop.hbase.security.access.AccessController,
    org.apache.hadoop.hbase.security.token.TokenProvider
  </value>
</property>
<property>
  <name>hbase.coprocessor.master.classes</name>
  <value>
    org.apache.hadoop.hbase.security.access.AccessController
  </value>
</property>
<property>
  <name>hbase.coprocessor.regionserver.classes</name>
  <value>
    org.apache.hadoop.hbase.security.access.AccessController
  </value>
</property>
<property>
  <name>hbase.security.exec.permission.checks</name>
  <value>true</value>
</property>
```

(4) 创建一个 Kerberos 主体，执行 HBase 管理功能：

```
kadmin: addprinc hbase@EXAMPLE.COM
WARNING: no policy specified for hbase@EXAMPLE.COM; defaulting to no \
policy
Enter password for principal "hbase@EXAMPLE.COM":
Re-enter password for principal "hbase@EXAMPLE.COM":
Principal "hbase@EXAMPLE.COM" created.
kadmin:
```

(5) 为该应用创建一个 Kerberos 主体，并将密钥导出到 keytab 文件：

```
kadmin: addprinc web-page-snapshots@EXAMPLE.COM
WARNING: no policy specified for web-page-snapshots@EXAMPLE.COM;
defaulting to no policy
Enter password for principal "web-page-snapshots@EXAMPLE.COM":
Re-enter password for principal "web-page-snapshots@EXAMPLE.COM":
Principal "web-page-snapshots@EXAMPLE.COM" created.
kadmin: ktadd -k app.keytab web-page-snapshots
Entry for principal web-page-snapshots with kvno 4, encryption type
des3-cbc-sha1 added to keytab WRFILE:app.keytab.
Entry for principal web-page-snapshots with kvno 4, encryption type
arcfour-hmac added to keytab WRFILE:app.keytab.
Entry for principal web-page-snapshots with kvno 4, encryption type
des-hmac-sha1 added to keytab WRFILE:app.keytab.
Entry for principal web-page-snapshots with kvno 4, encryption type
des-cbc-md5 added to keytab WRFILE:app.keytab.
kadmin:
```

(6) 将 `keytab` 文件复制到该应用用户的 home 目录；

(7) 授予应用主体创建表的权限：

```
[app@snapshots ~]$ kinit hbase
Password for hbase@ENT.CLOUDERA.COM:
[app@snapshots ~]$ hbase shell
14/11/13 14:45:53 INFO Configuration.deprecation: hadoop.native.lib is
deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0> grant 'web-page-snapshots', 'RWXCA'
0 row(s) in 4.0340 seconds

hbase(main):002:0>
```

(8) 创建 HBase 表：

```
[app@snapshots ~]$ kinit -kt ~/app.keytab web-page-snapshots
[app@snapshots ~]$ export KITE_USER_CLASSPATH=/etc/hadoop/conf
[app@snapshots ~]$ export \
ZK=zk1.example.com,zk2.example.com,zk3.example.com
[app@snapshots ~]$ kite-dataset create \
  dataset:hbase:${ZK}:2181/webpagesnapshots.WebPageSnapshotModel \
  -s src/main/avro/hbase-models/WebPageSnapshotModel.avsc
[app@snapshots ~]$ kite-dataset create \
  dataset:hbase:${ZK}:2181/webpageredirects.WebPageRedirectModel \
  -s src/main/avro/hbase-models/WebPageRedirectModel.avsc
[app@snapshots ~]$
```

(9) 授予用户 alice 和 bob 访问公共表 / 列的权限：

```
[app@snapshots ~]$ kinit -kt ~/app.keytab web-page-snapshots
[app@snapshots ~]$ hbase shell
14/11/13 14:45:53 INFO Configuration.deprecation: hadoop.native.lib is
deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0> grant 'alice', 'RW', 'webpagesnapshots', 'content',
'public'
0 row(s) in 2.9580 seconds

hbase(main):002:0> grant 'alice', 'RW', 'webpagesnapshots', '_s'
0 row(s) in 0.1640 seconds

hbase(main):003:0> grant 'alice', 'RW', 'webpagesnapshots', 'meta'
0 row(s) in 0.2100 seconds

hbase(main):004:0> grant 'alice', 'RW', 'webpagesnapshots', 'observable'
0 row(s) in 0.1600 seconds

hbase(main):005:0> grant 'alice', 'RW', 'webpageredirects'
0 row(s) in 0.1600 seconds

hbase(main):006:0> grant 'alice', 'RW', 'managed_schemas'
```

```
0 row(s) in 0.1570 seconds

hbase(main):007:0> grant 'bob', 'RW', 'webpagesnapshots', 'content',
'public'
0 row(s) in 0.1920 seconds

hbase(main):008:0> grant 'bob', 'RW', 'webpagesnapshots', '_s'
0 row(s) in 0.1510 seconds

hbase(main):009:0> grant 'bob', 'RW', 'webpagesnapshots', 'meta'
0 row(s) in 0.2100 seconds

hbase(main):010:0> grant 'bob', 'RW', 'webpagesnapshots', 'observable'
0 row(s) in 0.1640 seconds

hbase(main):011:0> grant 'bob', 'RW', 'webpageredirects'
0 row(s) in 0.1590 seconds

hbase(main):012:0> grant 'bob', 'RW', 'managed_schemas'
0 row(s) in 0.1870 seconds

hbase(main):013:0>
```

(10) 授予 alice 和 bob 访问各自私有列的权限：

```
[app@snapshots ~]$ kinit -kt ~/app.keytab web-page-snapshots
[app@snapshots ~]$ hbase shell
14/11/13 14:45:53 INFO Configuration.deprecation: hadoop.native.lib is
deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0> grant 'alice', 'RW', 'webpagesnapshots', 'content',
'alice'

0 row(s) in 2.8890 seconds
hbase(main):002:0> grant 'bob', 'RW', 'webpagesnapshots', 'content',
'bob'
0 row(s) in 0.1600 seconds

hbase(main):003:0>
```

(11) 添加如下参数到所有 HBase 节点的 hbase-site.xml 文件，从而启用利用 web-page-snapshots 主体的用户模拟：

```
<property>
  <name>hadoop.proxyuser.web-page-snapshots.groups</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.web-page-snapshots.hosts</name>
  <value>*</value>
</property>
```



还有一些本示例应用的设计和实现所特有的其他配置步骤，运行该示例的完整步骤可参见 Github 上本项目的 README 文档（https://github.com/hadoop-security/kite-spring-habse-example）。

13.2.4 实现中的注意事项

为应用添加安全特性时，我们做了一些实现上的修改。虽然可以通过比较我们的示例与原始的 Kite SDK 示例查看完整的修改列表，但此处依然总结一下关键的修改。第一处修改是增加了额外的 Kerberos 登录模块，以通过应用的 keytab 获取 Kerberos TGT，Spring 初始化其他 Web 应用之前会加载该模块。下面是该模块的一个不包括登录或错误检查的简化版：

```
public class KerberosLoginService {

  public KerberosLoginService(String applicationPrincipal,
      String applicationKeytab) throws IOException {

    if (UserGroupInformation.isSecurityEnabled()) {
      UserGroupInformation.loginUserFromKeytab(applicationPrincipal,
          applicationKeytab);
    }
  }
}
```

这里的要点是，我们使用 `UserGroupInformation` 类的 `loginUserFromKeytab()` 方法前，要首先确认已经启用了集群上的安全策略。该方法会使用 keytab 文件获取我们的 Kerberos TGT。

从 Hadoop 安全角度考虑的第二处所需的改动是，修改 `WebPageSnapshotService` 以模拟认证后的用户与 HBase 通信。为了实现这个目标，使用 `UserGroupInformation` 对象的 `doAs()` 方法，该对象代表了我们想要模拟的代理用户。如下示例向 `WebPageSnapshotService` 中的一个方法添加身份模拟：

```
private WebPageSnapshotModel getWebPageSnapshot(String url,
    final long ts, final String user) throws IOException {
  WebPageSnapshotModel snapshot = null;
  final String normalizedUrl = normalizeUrl(url, user);

  UserGroupInformation ugi = UserGroupInformation.createProxyUser(user,
      UserGroupInformation.getLoginUser());
  snapshot = ugi.doAs(new PrivilegedAction<WebPageSnapshotModel>() {

    @Override
    public WebPageSnapshotModel run() {
      Key key = new Key.Builder(webPageSnapshotModels(user))
          .add("url", normalizedUrl)
          .add("fetchedAtRevTs", Long.MAX_VALUE - ts).build();
      return webPageSnapshotModels(user).get(key);
    }
```

```
  });

  return snapshot;
}
```

最后一个所需的改动是，从使用单一共享的 HBase 连接变成为每个用户创建一个连接，这是由 HBase 客户端缓存连接的方法决定的。最重要的关键点是，为每个用户创建连接，并在 HBase 要使用的 Configuration 对象中，将 hbase.client.instance.id 设为唯一值。对于该应用，我们创建一个实用方法以创建和缓存连接：

```
private synchronized RandomAccessDataset<WebPageSnapshotModel>
  webPageSnapshotModels(String user) {

  RandomAccessDataset<WebPageSnapshotModel> dataset =
      webPageSnapshotModelMap.get(user);

  if (dataset == null) {
    Configuration conf = new Configuration(
        DefaultConfiguration.get());
    conf.set("hbase.client.instance.id", user);
    DefaultConfiguration.set(conf);
    dataset = Datasets.load(webPageSnapshotUri,
        WebPageSnapshotModel.class);
    webPageSnapshotModelMap.put(user, dataset);
  }

  return dataset;
}
```

13.2.5 小结

本案例分析中，我们对一个典型交互式 HBase 应用的设计和架构进行了综述，然后探讨了与用例相关的安全考虑（身份验证、授权、身份模拟等），还描述了要支持我们的授权模型所必需的数据模型改动。接下来，总结了想要增加到应用中的安全需求，并给出配置集群，以满足该安全需求的必要步骤。最后，描述了应用实现中要支持这些安全需求所需的部分修改。

后记

问世以来，Hadoop 已经走过很长一段路。正如你在本书中看到的，安全在整个生态系统中包含大量内容。随着大数据的兴起，以及在那些迅速使用 Hadoop 作为数据平台的企业中的影响，Hadoop 及其庞大生态系统的快速发展已不足为奇。尽管如此，Hadoop 仍处于非常早期的阶段。虽然有很多可用的安全配置，但 Hadoop 要想达到关系型数据库和数据仓库的水准，以完全满足资产上亿的公司在数据管理上的需求，还有很多工作要做。

好在由于 Hadoop 在市场上的巨大增长，人们在快速填补其安全空缺。书中没有讨论的要么是现在正处于开发阶段（可能在本书出版时已经完成）的内容，要么是在不久的将来会变成 Hadoop 生态系统一部分的特性。

统一授权

Hadoop 安全管理员最困难的工作之一就是，要跟踪无数组件是如何处理访问控制的。虽然我们把很多东西加到 Apache Sentry，以作为 Hadoop 的集中授权组件，但在为整个生态系统提供授权方面还不够。从长期来看这是迟早的，也是必需的。安全管理员和审计员都需要有一个地方供他们查看和管理所有用户授权控制相关的策略，如果缺少这个，就很容易在过程中犯错。

短期而言，Apache Sentry 将会集成 HDFS 授权，这将允许有一个统一的方法定义在组件之间共享数据时的数据访问策略。例如，如果数据被载入 Hive 仓库并由 Sentry 策略控制，那将如何处理 MapReduce 访问？正如第 13 章所述，这涉及使用扩展 HDFS ACL。有了与 Sentry 集成的 HDFS 后，HDFS 路径可以被指定为由 Sentry 控制。因此，授权决策取决于 Sentry 策略，而不是标准 POSIX 权限或者扩展 ACL。

另外，Sentry 即将与 HBase 集成。我们在第 6 章看到，授权策略存储在 HBase 的一个特殊表中，并在默认情况下通过 HBase 命令行进行管理。将策略存储迁移到 Sentry 是一个好的选择。

数据管控

本书不讨论数据管控的大型话题，但讨论了其中一个关于审计的子话题。如第 8 章所述，集群中很多捕捉事件的地方都有审计日志，但没有集中的地方全面捕捉审计，也没有地方进行常规的数据管控任务，如管理业务元数据、查看联系和族谱、管理数据留存等。传统数据仓库中都突出包含了这些功能，为了使 Hadoop 能够在整体安全上更进一步，数据治理需要得到比现在更好的解决。

原生数据保护

除了加密之外，Hadoop 还需要遮蔽和标记化的原生方法。虽然遮蔽可以创造性地通过 UDF 或者专门的视图实现，但提供基于预设策略遮蔽在线数据的功能则会更有意义。目前有其他商业产品提供这个功能，但我们相信，Hadoop 需要一个这样的功能作为其原生功能。当前，不使用商业产品的条件下不可能支持标记化。然而标记化对于数据科学家而言很重要，尤其是因为他们可能不需要看数据的具体值，但需要保留数据的关联关系和其他静态属性以进行分析。这不可能用掩蔽实现，但可以用标记化实现。

结语

Hadoop 和大数据是令人兴奋的市场。虽然一些安全专业人员，尤其是习惯了更加统一安全功能的安全专业人员可能会对它有一些望而却步，但我们希望本书能够帮助读者理解 Hadoop 安全的现状，并体会到，即便是具有很多组件的庞大 Hadoop 集群，也能使用精心策划的安全架构进行保护。

关于作者

Ben Spivey 目前是 Cloudera 的一名解决方案架构师，负责为客户在 Hadoop 部署方面提供咨询。Ben 曾在多家世界 500 强企业工作，涉及金融服务、零售、医疗保健等多个行业。他的专长在于对客户的 Hadoop 集群进行规划、安装、配置以及安全保护。

在 Cloudera 之前，Ben 与某个国防承包商一起为美国国家安全局（NSA）工作。在此期间，Ben 的工作之一就是建立一系列应用，这些应用被集成到企业安全基础设施中以保护敏感信息。

Joey Echeverria 是 Rocana 的一位软件工程师，其工作是在 Apache Hadoop 平台建立下一代 IT 运行分析系统。Joey 还是一位 Kite SDK 贡献者，Kite SDK 是一个 Apache 许可的 Hadoop 生态系统数据 API。Joey 之前是 Cloudera 的软件工程师，在 Cloudera 期间，他为 Apache Flume、Apache Sqoop、Apache Hadoop、Apache HBase 等众多 ASF 项目做出了贡献。

关于封面

本书封面上的动物是日本獾（学名 **Meles anakuma**），鼬科。顾名思义，它是日本特有的动物，生活于本州、九州、四国和小豆岛。

与欧洲同类相比，日本獾体型较小，雄性体长约 31 英寸（79 厘米），雌性略小，平均为 28 英寸（71 厘米）。除了犬齿大小之外，雄性和雌性在体格上没有多大区别。成年日本獾的体重为 8.8 磅（4 千克）~17.6 磅（8 千克），躯干粗钝，四肢短小。其前肢拥有强大的挖土爪，而后肢较小。虽然没有欧洲獾那样独特，不过日本獾的特点是，面部有黑白相间的条纹。

日本獾在夜间活动，冬天冬眠。雌性獾 2 岁时，就会在春天求偶，并生育 2~3 只幼獾。与欧洲同类相比，日本獾更习惯独处，没有长期的配偶关系。

日本獾居住在各种树林和森林栖息地，是杂食性动物，以虫子、甲虫、浆果和柿子为食。

O’Reilly 图书封面上的很多动物都是濒危动物，它们对世界都很重要。要想详细了解如何帮助这些动物，请访问 animals.oreilly.com。

封面图片来源未知。

O'REILLY®

Hadoop安全：大数据平台隐私保护

随着使用Hadoop存储并处理大量数据的企业不断增多，Hadoop安全性日益凸显，尤其是在金融和医疗等涉及敏感信息的行业。本书两位作者均来自Hadoop安全防范一线，书中详细论述了身份验证、加密、密钥管理等诸多重要主题，并给出了具体处理建议和案例分析，读者可以从中了解搭建和使用Hadoop的资深开发者们是如何安全管理大数据的。

- 了解分布式系统，尤其是Hadoop所面临的安全挑战
- 学习如何确保Hadoop集群硬件的安全性
- Kerberos网络认证协议概览
- 身份验证、授权和审计原则在Hadoop中的应用
- 静态数据和动态数据的加密
- 客户端访问和数据提取过程的安全防护措施

Ben Spivey，Cloudera解决方案架构师，曾在多家世界500强企业工作，涉及金融服务、零售、医疗等多个行业，在对客户的Hadoop集群进行规划、安装、配置和安全防护方面有丰富经验。

Joey Echeverria，Rocana软件工程师，负责在Hadoop平台下构建下一代IT运行分析系统。Hadoop生态数据系统API Kite SDK的贡献者，并为Flume、Hadoop、HBase等多个Apache项目做过贡献。

“Hadoop能够让你存储更多数据，并使用多种高效工具对其进行挖掘。本书帮你了解如何安全无忧地体验Hadoop这些强大性能。”

——Doug Cutting
Hadoop之父

“本书的两位作者在将安全概念引入Hadoop平台方面做出过突出贡献，他们不但介绍了Hadoop从早期开放的消费互联网时代到现在作为敏感数据可信平台的演变过程，还对如何安全管理大数据给出了具体意见。”

——Mike Olson
Cloudera公司首席战略官、联合创始人

DATA

封面设计：Ellie Volkhausen 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机 / 大数据 / Hadoop

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版





ISBN 978-7-115-46771-3
定价：79.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks